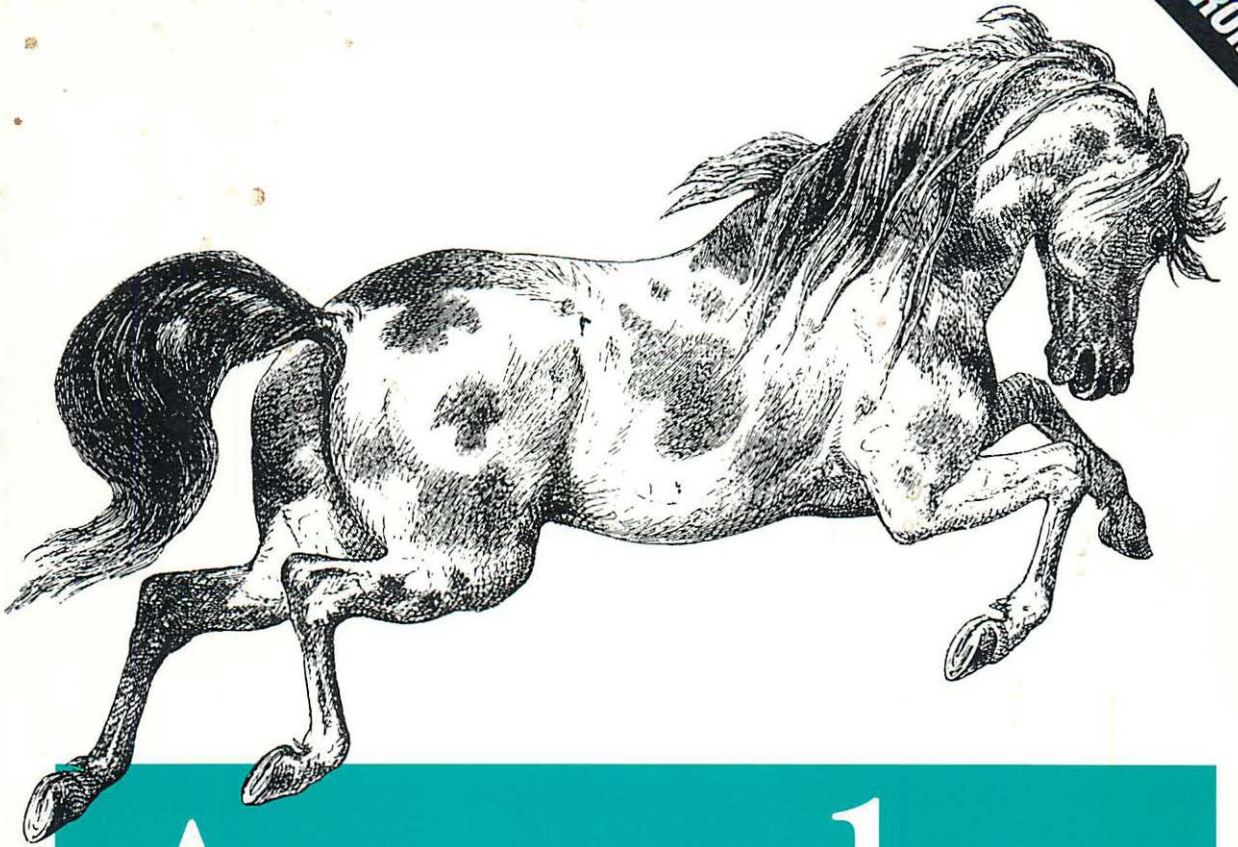


*Vital Information for Apache
Programmers and Administrators*

 Includes
CD-ROM



Apache

The Definitive Guide

A NUTSHELL®



HANDBOOK

Ben Laurie & Peter Laurie

O'Reilly & Associates, Inc.

001

GOOGLE EXHIBIT 1029

Apache
The Definitive Guide

Apache

The Definitive Guide

Ben Laurie and Peter Laurie

O'REILLY™
Cambridge • Köln • Paris • Sebastopol • Tokyo

Apache: The Definitive Guide

by Ben Laurie and Peter Laurie

Copyright © 1997 Ben Laurie and Peter Laurie. All rights reserved.

Printed in the United States of America.

Published by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.

Editor: Robert Denn

Production Editor: Mary Anne Weeks Mayo

Printing History:

March 1997: First Edition.

Nutshell Handbook and the Nutshell Handbook logo are registered trademarks of O'Reilly & Associates, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly & Associates, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.



This book is printed on acid-free paper with 85% recycled content, 15% post-consumer waste. O'Reilly & Associates is committed to using paper with the highest recycled content available consistent with high quality.

ISBN: 1-56592-250-6

[7/97]

Table of Contents

<i>Preface</i>	<i>ix</i>
1. <i>Getting Started</i>	1
How Does Apache Work?	4
What to Know About TCP/IP	5
What Is HTTP?	7
How Does Apache Use TCP/IP?	8
Hostnames	10
What the Client Does	10
What Happens at the Server End?	12
Compiling Apache	12
Which Apache?	13
Which UNIX?	13
gzip	13
Making Apache	15
Installing Apache	19
2. <i>Our First Web Site</i>	20
WebUser and Webgroup	22
httpd's Flags	25
Permissions	25
A Local Network	27
An Experimental Web Site	27
site.simple	29

3. <i>Towards a Real Web Site</i>	32
Butterthlies, Inc.	32
Default Index	34
Index.html	34
Directives	35
Block Directives	38
Two Copies of Apache	39
Virtual Hosts	41
Controlling Virtual Hosts	43
Restarts	45
HTTP/1.1 Virtual Hosts	46
Using .htaccess Files	47
4. <i>Common Gateway Interface (CGI)</i>	53
Directives	55
Editing the Brochure	55
echo.c	59
Setting Environment Variables	61
Browsers	62
suEXEC	63
HTTP Response Headers	64
Handlers	67
Actions	68
5. <i>Authentication</i>	70
Directives	71
Passwords	73
New Order Form	75
Automatic User Information	77
Digest Authentication	82
Anonymous Access	85
A Bad Mistake You Might Prefer to Avoid	88
6. <i>Content and Language Arbitration</i>	90
Image Negotiation	91
Language Negotiation	91
Type Maps	93
Browsers and HTTP/1.1	95
Expirations	96

7. Indexing	98
Making Better Indexes in Apache	98
Making Our Own Indexes	104
8. Redirection	107
ScriptAlias	108
Alias	108
Redirect	109
Rewrite	109
Image Maps	116
9. Proxy Server	122
Proxy Directives	122
Caching	124
Setup	126
10. Server-Side Includes	129
File Size	132
File Modification Time	132
Includes	133
Execute CGI	133
Echo	134
XBitHack	134
XSSI	134
11. What's Going On?	136
Server Status	136
Server Info	138
Logging the Action	143
12. Extra Modules	148
Authentication	149
Blocking Access	149
Counters	149
Faster CGI Programs	150
Languages and Internationalization	150
Miscellaneous	150
Server-Side Scripting	150
Throttling Connections	150
URL Rewriting	150

13. The Apache API	152
Pools	152
Per-Server Configuration	153
Per-Directory Configuration	154
Per-Request Information	154
Access to Configuration and Request Information	156
Functions	156
14. Writing Apache Modules	175
Overview	175
Status Codes	176
The Module Structure	178
A Complete Example	197
General Hints	204
15. Security	205
Internal and External Users	206
Apache's Security Precautions	208
Validating Users	209
Binary Signatures, Virtual Cash	209
Firewalls	214
Legal Issues	218
Secure Sockets Layer: How To Do It	222
Apache-SSL's Directives	229
Cipher Suites	231
A. Support Organizations	233
B. Echo Program	235
C. NCSA and Apache Compatibility	239
D. SSL Protocol	241
Index	247

Preface

This book is principally about the Apache web server software. We explain below what a web server is and how it works, but our assumption is that most people who read this book will have used the World Wide Web and understand in practical terms how it works, and that they are now thinking about running their own server to offer material to the hungry masses.

This book takes the reader through the process of acquiring, compiling, installing, configuring, and modifying Apache. In the process, we exercise most of the package's functions. We do this through a set of example sites that take a reasonably typical web business—in our case a postcard publisher—through a process of development and increasing complexity. However, we have deliberately not tried to make each site more complicated than the last. Most of the chapters refer to an illustrative site that is as simple as we could make it. Each site is pretty well self-contained so that the reader can refer to it to follow the text without having to disentangle the meat there from extraneous vegetables. If desired, it is perfectly possible to install and run each site on a suitable system.

Apache is a versatile package and getting more so every day, so we have not tried to illustrate every possible combination of commands. That would require a book of a million pages or so. Rather, we have tried to suggest lines of development that a typical webmaster should be able to follow once an understanding of the basic concepts is clear.

Writing the book was something of a race with Apache's developers. Version 1.1.1 was released just as we started, and then work began on the next version, which was enhanced almost daily as we wrote. In some cases we had to tear up earlier material because new releases changed the functionality under our feet. When we finished, version 1.2 was in development though not yet stabilized. We have tried to deal with its features, but some are still in debate.

In many of the examples that follow, the motivation for what we make Apache do is simple enough and requires little explanation (for example, the different index formats in Chapter 7, *Indexing*). Elsewhere we feel that the webmaster needs to be aware of wider issues before making sensible decisions about his or her site's configuration (for instance, security issues in Chapter 15, *Security*), and we have not hesitated to branch out to deal with them.

Who Wrote Apache and Why?

Apache gets its name from the fact that it consists of some existing code plus some patches. The FAQ* thinks that this is cute; others may think it's the sort of joke that gets programmers a bad name. A more responsible group thinks that Apache is an appropriate name because of the resourcefulness and adaptability of the American Indian tribe.

You have to understand that Apache is free to its users and is written by a voluntary team who do not get paid for their work. Whether or not they decide to incorporate your, or anyone else's, ideas is entirely up to them. If you don't like this, feel free to start collecting a team and write your own web server.

The first web server was built by Tim Berners-Lee at CERN, the European Centre for Nuclear Research (yes, the initials are not in the same order, but CERN's real name is in French), at Geneva, Switzerland. The immediate ancestor of Apache was built by the U.S. government, in the person of NCSA, the National Center for Supercomputing Applications. This fine body is not to be confused with the National Computing Security Agency or the North Carolina Schools Association. Since this code was written with (American) tax payers' money, it is available to all and you can, if you like, download the source code in C from www.ncsa.uiuc.edu, paying due attention to the license conditions.†

There were those who thought that things could be done better, and in the FAQ for Apache (at <http://www.apache.org>) we read:

...Apache was originally based on code and ideas found in the most popular HTTP server of the time, NCSA httpd 1.3 (early 1995).

That phrase: "of the time" is nice. It usually refers to good times back in the 1700s or the early days of technology in the 1900s. But now it means back in the deliquescent bogs of a year ago!

* FAQ is netspeak for Frequently Asked Questions. Most sites/subjects have an FAQ file that tells you what the thing is, why it is, and where it is going. It is perfectly reasonable for the newcomer to ask for the FAQ to look up anything new to him or her, and indeed a sensible thing to do since it reduces the number of questions asked. Apache's FAQ can be found at <http://www.apache.org/docs/FAQ.html>.

† There were rumors as we went to press that NCSA might end development of their server.

While the Apache site is open to all, Apache is written by an invited group of (we hope) reasonably good programmers who give their time to the project for free. One of the authors of this book (Ben) is a member of this group.

Why do they bother? Why do these programmers, who presumably could be well paid for doing something else, sit up nights to work on Apache for our benefit? There is no such thing as a free lunch, and they do it for a number of typically human reasons. One might list, in no particular order:

- They want to do something more interesting than their day job, which might be writing stock-control packages for BigBins, Inc.
- They want to be involved on the edge of what is happening. Working on a project like this is a pretty good way to keep up-to-date. After that comes consultancy on the next hot project.
- The more worldly ones might remember how, back in the old days of 1995, quite a lot of the people working on the web server at NCSA left for a thing called Netscape and became, in the passage of the age, zillionaires.
- It's fun. Developing good software is interesting and amusing and you get to meet and work with other clever people.
- They are not doing the bit that programmers hate: explaining to end users why their treasure isn't working and trying to fix it in 10 minutes flat. If you want support on Apache you have to consult one of several commercial organizations (see Appendix A, *Support Organizations*) who, quite properly, want to be paid for doing the work everyone loathes.

The Demonstration CD-ROM

The CD-ROM that accompanies this book contains the requisite *README* file with installation instructions and other useful information. It also contains a number of demonstration web sites. Most of the chapters that follow refer to one or more of these sites. These sites and some other useful directories are in the directory *.../www* that contains:

<code>cgi-bin</code>	<i>Home of the CGI scripts</i>
<code>lan_setup</code>	<i>Script to set up demonstration network</i>
<code>main_docs</code>	<i>Commonly used HTML scripts</i>
<code>ok_dbm</code>	<i>DBM passwords</i>
<code>ok_digest</code>	<i>Digest passwords</i>
<code>ok_users</code>	<i>Ordinary passwords</i>
<code>site.alias</code>	<i>Demonstration sites</i>
<code>site.anon</code>	
<code>site.authent</code>	
<code>site.cgi</code>	
<code>site.digest</code>	

site.fancyindex	
site.filter	
site.first	
site.htaccess	
site.imap	
site.logging	
site.multiview	
site.names	
site.ownindex	
site.proxy	
site.refer	
site.rewrite	
site.simple	
site.ssi	
site.ssl	
site.status	
site.toddle	
site.twocopy	
site.typemap	
site.virtual	
somewhere_else	<i>"Remote" site</i>
tmp	<i>Temporary storage</i>

In addition to the demonstration web sites, we've included other useful files:

apache_1.1.1.tar

Apache Version 1.1.1

apache_1.2b.tar

Apache Version 1.2b, including the complete online Apache documentation

rfc2068.txt

The HTTP/1.1 specification

rfc1945.txt

The HTTP/1.0 specification

draft-robinson-www-interface-01.txt

The CGI/1.1 specification

gzip

gzip Version 1.2.4

patch

patch Version 2.1

Conventions Used in This Book

This section covers the different conventions used in this book.

Typographic Conventions

ConstantWidth

Used for HTTP headers, status codes, MIME content types, directives in configuration files, commands, options/switches, keys, buttons, menu items, and computer output in text

ConstantWidth bold

Used for input to be typed in by the user

Italic

Used for filenames, pathnames, newsgroup names, Internet addresses (URLs), email addresses, variable names (except in examples), terms being introduced, program names, subroutine names, functions, arrays, operators, methods, and hostnames

ALL CAPS

Used for environment variables, protocols, HTML attributes, and HTML tags (within angle brackets <>)

Pathnames

We use the text convention `.../` to indicate your path to the demonstration sites, which may well be different from ours. For instance, on our Apache machine, we kept all the demonstration sites in the directory `/usr/www`. So, for example, we had `/usr/www/site.simple`. You might want to keep them somewhere other than `/usr/www`, so we refer to the path as `.../site.simple`.

Don't type `.../` into your computer. The attempt will upset it!

Directives

Apache is controlled through roughly 150 directives. Each directive has a formal explanation in the following format:

Directive

Syntax

Where used

Explanation

So, for instance, we have:

ServerAdmin

ServerAdmin email address
server config, virtual host

ServerAdmin gives the *email address* for correspondence. It automatically generates error messages so the user has someone to write to in case of problems.

The “where used” line explains the appropriate environment for the directive. This will become clearer later.

Organization of This Book

The chapters that follow and their contents are:

Chapter 1, Getting Started

Covers web servers, how Apache works, TCP/IP, HTTP, hostnames, what a client does, what happens at the server end, choosing a UNIX, and compiling and installing Apache.

Chapter 2, Our First Web Site

Discusses getting Apache to run, creating Apache users, run-time flags, permissions, and site.simple.

Chapter 3, Towards a Real Web Site

Introduces a demonstration business, Butterthlies, Inc.; some HTML; default indexing of web pages; server housekeeping; and block directives.

Chapter 4, Common Gateway Interface (CGI)

Demonstrates aliases, logs, HTML forms, shell script, a CGI in C, environment variables, and adapting to the client's browser.

Chapter 5, Authentication

Explains controlling access, collecting information about clients, cookies, DBM control, digest authentication, and anonymous access.

Chapter 6, Content and Language Arbitration

Covers content and language arbitration, type maps, and expiration of information.

Chapter 7, Indexing

Discusses better indexes, index options, your own indexes, and image maps.

Chapter 8, Redirection

Describes `Alias` and `ScriptAlias`, the amazing rewrite module.

Chapter 9, Proxy Server

Covers remote proxy and proxy caching.

Chapter 10, Server-Side Includes

Explains run-time commands in your HTML and XSSI—a more secure SSI.

Chapter 11, What's Going On?

Covers server status, logging the action, and configuring the log files.

Chapter 12, Extra Modules

Discusses authentication, blocking, counters, faster CGI, languages, server-side scripting, and URL rewriting.

Chapter 13, The Apache API

Describes pools, per server/per directory/per request information, functions, warnings, example, and parsing.

Chapter 14, Writing Apache Modules

Covers status codes, module structure, command table, initializer, translate name, check access, check userid, check authorization, check type, pre-run fixups, handlers, logger, and a complete example.

Chapter 15, Security

Discusses Apache's security precautions, validating users, binary signatures, virtual cash, certificates, firewalls, packet filtering, secure sockets layer (SSL), legal issues, patent rights, national security, SSL: how to do it, and Apache-SSL directives.

Appendix A, Support Organizations

Provides a list of commercial service and/or consultation providers.

Appendix B, Echo Program

Provides a listing of *echo.c*.

Appendix C, NCSA and Apache Compatibility

Contains Apache group internal mail discussing NCSA/Apache compatibility issues.

Appendix D, SSL Protocol

Provides the SSL specification.

Acknowledgments

First, thanks to Robert S. Thau who gave the world the Apache API and the code that implements it, and to the Apache Group who have worked on it before and since. Thanks to Eric Young and Tim Hudson for giving SSLey to the Web.

Thanks to Bryan Blank, Aram Mirzadeh, Chuck Murcko, and Randy Terbush who read early drafts of the text and made many useful suggestions, and to Jessica Perry Hekman who reviewed the CD-ROM. Thanks to Paul C. Kocher for allowing us to reproduce SSL Protocol, Version 3.0, in Appendix D, *SSL Protocol*, and to

Netscape Corporation for allowing us to reproduce *echo.c* in Appendix B, *Echo Program*.

Many thanks to Robert Denn, our editor at O'Reilly, who patiently turned our text into a book. The blunders that remain are our own contribution.

Many people in O'Reilly's production and design groups contributed to our project. Mary Anne Weeks Mayo was the production project manager and copy editor. Chris Reilley was responsible for the figures. Seth Maislin wrote the index. Quality control was assured by Sheryl Avruch and Ellie Fountain Maden. Mike Sierra contributed his FrameMaker tool-tweaking prowess. The book's interior was designed by Nancy Priest, and Edie Freedman designed the front and back covers.

And finally, thanks to Camilla von Massenbach and Barbara Laurie for putting up with us while we wrote this book.

In this chapter:

- *How Does Apache Work?*
- *What to Know About TCP/IP*
- *What Is HTTP?*
- *How Does Apache Use TCP/IP?*
- *Hostnames*
- *What the Client Does*
- *What Happens at the Server End?*
- *Compiling Apache*
- *Which Apache?*
- *Which UNIX?*
- *gzip*
- *Making Apache*
- *Installing Apache*

1

Getting Started

When you connect to the URL of someone's home page—say the notional *http://www.butterflies.com/* we shall meet later on—you send a message across the Internet to the machine at that address. That machine, you hope, is up and running, its Internet connection is working, and it is ready to receive and act on your message.

URL stands for Universal Resource Locator. Many people think that a complete web address is a URL. But (we paraphrase an email from Roy Fielding, one of the authors of the HTTP/1.1 RFC):

The terminology of URI, URLs, etc. seems to be not 100% clear. OK, the URI Working Group gave a definition. In principle: “URIs are the superclass of URLs and URNs.” And they gave some examples of what should be called URLs and what else are URNs.

But in the Web community, they are not used in this way: Often *http://www.anyhost.dom/foo/bar/quux/* is called a URI [Universal Resource Indicator], and its */foo/bar/quux/* part is called a URL. No, both are URLs (being locations), and both are URIs (since all URLs are URIs, by definition). The latter is called a URL relative to the server root, or an absolute path URL. You can see such references in the Relative URL spec, RFC 1808.

Use of URL and URI is not consistent because of politics—there was a period of about a year in which the URI WG managed to break all usage of WWW

addresses (URLs) by adding stupid requirements to their syntax. So, TimBL said to hell with that and changed WWW addresses to the name URI. Unfortunately, the NCSA documentation always used URLs, so now we are left in total confusion about what to name things. In the midst of that I wrote RFC 1808, which talks about relative URLs only (instead of relative URIs) simply because that was all we could get consensus on within that WG.

Later, when writing the HTTP spec, I decided that it would be silly to restrict HTTP to URLs simply because the URN people hadn't found a clue. So, I defined all of the HTTP stuff as using URIs instead of URLs.

In theory, this whole mess will straighten itself out when all the specs start moving to draft standards. I am supposed to write a revision of RFC 1808 at some point, but I'm really quite sick of the whole process right now.

A URL such as *http://www.butterthlies.com/* comes in three parts:

```
<method>://<host>/<absolute path URL (apURL)>
```

So, in the above example, *<method>* is *http*, meaning to use HTTP (Hypertext Transfer Protocol); *<host>* is *www.butterthlies.com* and *<apURL>* is *"/*, meaning the top directory of your host. Your browser might send the request:

```
GET / HTTP/1.0
```

It arrives at port 80 (the default HTTP port) on the host *www.butterthlies.com*. The message is again in three parts: a method (an HTTP method, not a URL method), in this case *GET*, but it could equally be *PUT*, *POST*, *DELETE*, or *CONNECT*; a URI, */*; and the version of the protocol we are using. (As we write, HTTP/1.1 is nearing release, but HTTP/1.0 is still the current standard.) It is then up to the web server running on that host to make something of this message.

It is worth saying here—and we will say it again—that the whole business of a web server is to translate a URL either into a filename and send that file back over the Internet; or into a program name, run that program, and send its output back. That is the meat of what it does: all the rest is trimming.

The host machine may be a whole cluster of hypercomputers costing an oil-sheik's ransom, or a humble PC. In either case, it had better be running a web server: a program that listens to the network, accepts, and acts on this sort of message by sending its organization's home page back to you. You read it and do whatever you do, and we'll look at that later.

What do we want a web server to do? It should:

- Run fast so it can cope with a lot of inquiries using a minimum of hardware.
- Be multitasking so it can deal with more than one inquiry at once.

- Be multitasking again so that the person running it can maintain the data it hands out without having to shut the service down. Multitasking is hard to arrange within a program: the only way to do it properly is to run the server on a multitasking operating system. In Apache's case this is UNIX, with Windows NT and OS2 coming along later.
- Authenticate inquirers: some may be entitled to more services than others. When we come to virtual cash, this feature (see Chapter 15, *Security*) becomes essential.
- Respond to errors in the messages it gets with answers that make sense in the context of what is going on. For instance, if a client requests a page the server cannot find, the server should respond with a "404" error, which is defined by the HTTP specification to mean "page does not exist."
- Negotiate a style and language of response with the inquirer. For instance, it should—if the people running the server can rise to the challenge—be able to respond in the language of the inquirer's choice. This, of course, can open out a lot more action to your site. And there are parts of the world where a response in the wrong language can be a bad thing to do. If you were operating in Canada where the English/French divide arouses bitter feelings, or in Belgium where the French/Flemish split is as bad, this feature could make or break your business.
- Offer different formats. On a more technical level, a user might want JPEG image files rather than GIF, or TIFF rather than either. He might want text in vdi format rather than PostScript.
- Run as a proxy server. A proxy server is a server that accepts requests for clients and forwards them to the real servers, and then sends the real servers' responses back to the clients. There are two reasons why you might want it to do this:
 - The proxy might be running on the far side of a firewall (see Chapter 15) giving its users access to the Internet.
 - It might cache popular pages to save reaccessing them.
- Be secure. The Internet world is like the real world, peopled by a lot of lambs and a few wolves.* The wolves like to get into the lambs' folds (of which your computer is one) and when there, raven and tear in the usual wolfish way. The aim of a good server is to prevent this happening. The subject of security is so important that we will come back to it several times before we are through.

* We generally follow the convention of calling these people the Bad Guys. This avoids debate about hackers, which, to many people, simply refers to good programmers, but to some means Bad Guys.

These are services the developers of Apache think a server should offer. There are people who have other ideas, and as with all software development, there are lots of features that might be nice: features someone might use one day, that might, if put into the code, actually make it work better instead of fouling up something else which has, until then, worked fine. Unless developers are careful, good software attracts so many improvements that it eventually rolls over and sinks like a ship caught in an Arctic ice storm.

Some ideas are in progress. For example, someone is toying with adding Python (a powerful scripting language) to avoid the overhead of CGI startup (see Chapter 4, *Common Gateway Interface (CGI)*). Someone else is playing with nonpreemptive multithreading (which may be useful for HTTP/1.1 as well as possibly improving performance).

If you have bugs to report or more ideas for development, send them to mail.apache_bugs@mail.apache.org. You can also try news:comp.infosystems.www.servers.unix where some of the Apache team as well as many other knowledgeable people lurk.

How Does Apache Work?

Apache is a program that runs under a suitable multitasking operating system. In the examples that follow in this book, the operating system is UNIX, but OS/2 and Windows 95 are equally possible. The binary is called *httpd* and normally runs in the background. Each copy of *httpd* that is started has its attention directed at a *web site*, which is, for practical purposes, a directory. For an example, look at *site.toddle* on the demonstration CD-ROM. A site directory typically contains three subdirectories:

conf

Contains the configuration file(s), of which *httpd.conf* is the most important. It is referred to throughout this book as the *Config file*.

htdocs

Contains the HTML scripts to be served up to the site's clients. This directory and those below it, the *web space*, are accessible to anyone on the web and therefore pose a severe security risk if used for anything other than public data.

logs

Contains the logging data, both of accesses and errors.

There is typically a directory outside the web space, usually called *cgi-bin*, where the CGI scripts live. These are programs or shell scripts written by or for the

webmaster that can be executed by Apache on behalf of its clients. It is most important, for security reasons, that this directory not be in the web space.

In its idling state, Apache does nothing but listen to the IP addresses and TCP port or ports specified in its Config file. When a request appears on a valid port, Apache receives the HTTP request and analyzes the headers. It then applies the rules it finds in the Config file and takes the appropriate action.

The webmaster's main control over Apache is through the Config file. He has some 150 *directives* at his disposal, and most of this book is an account of what they do and how to use them to reasonable advantage. The webmaster also has a couple of flags he can use when Apache starts up.

The webmaster also controls which *modules* are compiled into Apache. Each module provides the code to execute a number of directives. If there is a group of directives that aren't needed, the appropriate modules can be left out of the binary by commenting their names out in the *configuration file** that controls the compilation of the Apache sources. Discarding unwanted modules reduces the size of the binary and may improve performance.

Apache is *freeware*, and the intending user downloads the sources from www.apache.org or a suitable mirror site, compiles them, and runs the result. You can load the sources from the demonstration CD-ROM, but at the time this book was being written, the sources were changing daily as bugs were fixed and new features introduced. Although it sounds like a difficult business to download the sources, and configure and compile them, it only takes about 20 minutes, and is well worth the trouble.

What to Know About TCP/IP

In order to understand what follows in this book, you need to have a modest knowledge of what TCP/IP is and what it does. You'll find more than enough information in Craig Hunt's book on TCP/IP,[†] but what follows is, we think, what is necessary to know for this book's purposes.

What is TCP/IP? The initials stand for "Transmission Control Protocol/Internet Protocol." It is a set of protocols enabling computers to talk to each other over networks. The two protocols that give the suite its name are among the most important, but there are many others, and we shall meet some of them later. These protocols are embodied in programs on your computer written by

* It is important to distinguish between the configuration file used at compile time and the Config file used to control the operation of a web site.

† *TCP/IP Network Administration*, published by O'Reilly & Associates.

someone or other; it doesn't much matter who. TCP/IP seems unusual among computer standards in that the programs that implement them actually work, and their authors have not tried too much to improve the original conceptions.

TCP/IP only applies where there is a network. Each computer on a network that wants to use TCP/IP—and we haven't said what a network is yet—has a number called the *IP address* that looks like, for example, 192.168.123.1.

There are four parts separated by periods. Each one corresponds to a byte, so the whole thing is four bytes long, and you will, in consequence, seldom see any of the parts outside the range 0 through 255.

Although not required by protocol, by convention there is a dividing line somewhere inside this number: to the left is the network number and to the right, the host number.

Two machines on the same physical network (usually a local area network [LAN]) normally have the same network number and communicate directly using TCP/IP.

How do we know where the dividing the line is between network number and host number? The default dividing line is determined by the first of the four numbers; if the value of the first number is:

- 0 through 127 (first byte is 0xxxxxxx binary), the dividing line is after the first number, and this is a Class A network. There are few class A networks—125 usable ones—but each one supports many hosts: up to 16,777,214.
- 128 through 191 (first byte is 10xxxxxx binary), the dividing line is after the second number, and this is a Class B network. There are more class B networks—16,382—and each one can support up to 65,534 hosts.
- 192 through 223 (first byte is 110xxxxx binary), the dividing line is after the third number, and this is a Class C network. There are a huge number of class C networks—2,097,150—but each one supports a paltry 254 hosts.

The remaining values of the first number, 224 through 255, are not relevant here. Network numbers—the left-hand part—that are all 0s* or all 1s† in binary are reserved and therefore not relevant to us either. These are:

- 0.x.x.x
- 127.x.x.x
- 128.0.x.x

* An all-zero network address means “this network.” This is defined in STD 5 (RFC 791).

† An “all-one” network address means “broadcast.” This is also defined in STD 5 (RFC 922). In practice, broadcast network addresses are not very useful, and, indeed, some of these “reserved” addresses have already been used for other purposes, for example, 127.0.0.1 means “this machine,” by convention.

- 191.255.x.x
- 192.0.0.x
- 223.255.255.x

It is often possible to bypass the rules of class A, B, and C networks using *subnet masks*. These allow us to further subdivide the network by using more of the bits for the network number and less for the host number. Their correct use is rather technical, so we leave it to the experts.

You do not need to know this in order to run a host, because the numbers you deal with are assigned to you by your network administrator or are just facts of the Internet. But we feel you should have some understanding in order to avoid silly conversations with people who do know about TCP/IP. It is also relevant to virtual hosting because each virtual host (see Chapter 3, *Towards a Real Web Site*) must have its own IP address.*

Now we can think about how two machines with IP addresses X and Y talk to each other. If X and Y are on the same network and are correctly configured so that they have the same network number and different host numbers, they should be able to fire up TCP/IP and send packets to each other down their local, physical network without any further ado.

If the network numbers are not the same, TCP/IP sends the packets to a *router*, a special machine able, by processes that do not concern us here, to find out where the other machine is and deliver the packets to it. This may be over the Internet, but might be on your wide area network (WAN).

There are two ways computers use TCP/IP to communicate:

UDP (User Datagram Protocol)

A way to send a single packet from one machine to another. It does not guarantee delivery, and there is no acknowledgment of receipt. It is nasty for our purposes, and we don't use it.

TCP (Transmission Control Protocol)

A way to establish communications between two computers. It reliably delivers messages of any size. This is better.

What Is HTTP?

HTTP uses the TCP protocol. The client (which is normally a browser like Netscape) establishes a TCP connection to the server (which in our case is Apache) and then sends a request down that channel. The server examines the

* At least until HTTP/1.1 is in wide use.

request and responds in whatever way its webmaster has told it to. The webmaster does this by configuring his Apache server and the files or scripts he provides on the system. This book is mostly about this process.

The machine's response may be in Hypertext Markup Language (HTML), graphics, audio, VRML, or Java. Whatever it is in consists of bytes of data that are made into packets by the server's TCP/IP stack and transmitted. What those bytes are taken to mean at the other end is not really our problem.

How Does Apache Use TCP/IP?

Let's look at a server from the outside. We have a box in which there is a computer, software, and a connection to the outside world—a piece of Ethernet or a serial line to a modem or some such. This connection is known as an *interface* and is known to the world by its IP address. If the box had two interfaces, they would each have an IP address, and these addresses would normally be different. One interface, on the other hand, may have more than one IP address: see Chapter 3.

Requests arrive on an interface for a number of different services offered by the server using different protocols (the abbreviations in capitals):

- Network News Transfer Protocol (NNTP): news
- Simple Mail Transfer Protocol (SMTP): mail
- Domain Name Service (DNS)
- HTTP: World Wide Web

The server can decide how to handle these different requests because the four-byte IP address that leads the request to its interface is followed by a two-byte port number. Different services attach to different ports:

- NNTP: port number 119
- SMTP: port number 25
- DNS: port number 53
- HTTP: port number 80

You, as the local administrator or webmaster, can (if you really want) decide to attach any service to any port. Of course, if you decide to step outside convention, you need to make sure that your clients share your thinking. Our concern here is just with WWW and Apache. Apache, by default, listens to port number 80 because it deals in WWW business.

Port numbers below 1024 can only be used by the superuser (*root*, in other words); this prevents other users from running programs masquerading as standard services, but brings its own problems, as we shall see later.

This is fine if our machine is providing only one web server to the world. In real life you may want to host several, many, dozens of servers, which appear to the world to be completely different from each other. This situation was not anticipated by the authors of HTTP/1.0 (which is what we use now), and so handling a number of hosts on one machine has to be done by a kludge.

The kludge is that we assign multiple IP numbers to the same interface. By happy accident, the crucial UNIX utility, *ifconfig*, which binds IP addresses to physical interfaces, often allows the binding of multiple IP numbers so that people can switch from one IP number to another and maintain service during the transition.

In practical terms, on many versions of UNIX, we run *ifconfig* to give alias IP addresses to the same interface. The interface in this context is actually the bit of software—the driver—that handles the physical connection (Ethernet card, serial port, etc.) to the outside. While writing this book, we accessed the practice sites through an Ethernet connection between a Windows 95 machine (the client) and a FreeBSD box (the server) running Apache. The FreeBSD box was set up using *ifconfig* in the script *lan_setup*, which contained the following lines:

```
ifconfig ep0 192.168.123.2
ifconfig ep0 192.168.123.3 alias netmask 0xFFFFFFFF
ifconfig ep0 192.168.124.1 alias
```

The first line binds the IP address 192.168.123.2 to the physical interface *ep0*. The second binds an alias of 192.168.123.3 to the same interface. We used a subnet mask (*netmask 0xFFFFFFFF*) to suppress a tedious error message generated by the FreeBSD TCP/IP stack. This address was used to demonstrate virtual hosts. We also bound yet another IP address, 192.168.124.1, to the same interface, simulating a remote server in order to demonstrate Apache's proxy server. The important feature to note here is that the address 192.168.124.1 is on a different IP network from the address 192.168.123.2, even though it shares the same physical network. No subnet mask was needed in this case, as the error message it suppressed arose from the fact that 192.168.123.2 and 192.168.123.3 are on the same network.

Unfortunately, each UNIX tends to do this slightly differently, so these commands may not work on your system. Check your manuals!

Hostnames

In real life, we do not have much to do with IP addresses. Web sites (and Internet hosts generally) are known by their names—such as *www.butterflies.com*—which we shall meet later. On the authors' system, this name translates into 192.168.123.2. Another site on the same machine, *sales.butterflies.com*, translates into 192.168.123.3.

What the Client Does

Once this is set up, we can get down to business. The client has the easy end: it wants web action on a particular URL such as *http://www.apache.org*. What happens?

The browser observes that the URL starts with *http:* and deduces that it should be using the HTTP protocol. The *//* says that the URL is absolute, that is, not relative to some other URL. The next part must be the name of the server, *www.apache.org*. The client then contacts a nameserver, which uses DNS to resolve this name to an IP address. At the time of writing, this was 204.152.144.38. To see this process in action, go to the operating-system prompt and type:

```
% ping -c 5 www.apache.org
PING www.apache.org (204.152.144.38): 56 data bytes
64 bytes from taz.apache.org (204.152.144.38): icmp_seq=0 ttl=247
time=1380 ms
64 bytes from taz.apache.org (204.152.144.38): icmp_seq=1 ttl=247
time=1930 ms
64 bytes from taz.apache.org (204.152.144.38): icmp_seq=2 ttl=247
time=1380 ms
64 bytes from taz.apache.org (204.152.144.38): icmp_seq=3 ttl=247
time=1230 ms
64 bytes from taz.apache.org (204.152.144.38): icmp_seq=4 ttl=247
time=1360 ms
--- www.apache.org ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss round-trip
min/avg/max = 1230/1456/1930 ms
```

The web address *http://www.apache.org* doesn't include a port. This is because it is port 80, the default, and taken for granted. If some other port is wanted, it is included in the URL: for example, *http://www.apache.org:8000/*.

The URL always includes a path, even if is only */*. If the path is left out by the careless user, most browsers put it back in. If it were */some/where/foo.html* on port 8000, the URL would be *http://www.apache.org:8000/some/where/foo.html*.

The client now makes a TCP connection to port number 8000 on IP 204.152.144.38 and down the connection it sends the message:

```
GET /some/where/foo.html HTTP/1.0<CR><LF><CR><LF>
```

These carriage returns and line feeds (CRLF) are very important because they separate the HTML header from its body. If the request were a POST, there would be data following. The server sends the response back and closes the connection. To see it in action, connect again to the Internet, get a command-line prompt and type:

```
% telnet www.apache.org 80
```

telnet generally expects the host name followed by the port number. After connection, type:

```
GET /announcelist.html HTTP/1.0<CR><CR>
```

Since *telnet* also requires CRLF as the end of every line, it sends the right thing for you when you hit the Return key. Some implementations of *telnet* rather unnervingly don't echo what you type to the screen, so it seems that nothing is happening. But nevertheless, a whole mess of response streams past:

```
GET /announcelist.html HTTP/1.0
HTTP/1.1 200 OK
Date: Sun, 15 Dec 1996 13:45:40 GMT
Server: Apache/1.2b3-dev
Connection: close
Content-Type: text/html
Set-Cookie: Apache=arachnet784985065755545; path=/
<HTML>
<HEAD>
<TITLE>Join the Apache-Users Mailing List</TITLE>
</HEAD>
<BODY>
<IMG SRC="images/apache_sub.gif" ALT="">
<H1>Join the Apache-Announce Mailing List</H1>
<P>
The <code>apache-announce</code> mailing list has been set up to inform
people of new code releases, bug fixes, security fixes, and general
news and information about the Apache server. Most of this
information will also be posted to comp.infosystems.www.servers.unix,
but this provides a more timely way of accessing that information.
The mailing list is one-way, announcements only.
<P>
To subscribe, send a message to
<code><b>majorordomo@apache.org</b></code> with the words "subscribe
apache-announce" in the body of the message. Nope, we don't have a web
form for this because frankly we don't trust people to put the right
address. 
<A HREF="index"><IMG SRC="images/apache_home.gif" ALT="Home"></A>
</BODY></HTML>
Connection closed by foreign host.
```

What Happens at the Server End?

We assume that the server is well set up and running Apache. What does Apache do? In the simplest terms, it gets a URL from the Internet, turns it into a filename, and sends the file back down the Internet. That's all it does, and that's all this book is about!

Two main cases arise:

- The server has a standalone Apache that listens to one or more ports (port 80 by default) on one or more IP addresses mapped onto the interfaces of its machine. In this mode (known as *standalone mode*), Apache actually runs several copies of itself to handle multiple connections simultaneously.
- The server is configured to use the UNIX utility *inetd*. *inetd* listens on all ports it is configured to handle. When a connection comes in, it determines from its configuration file, */etc/inetd.conf*, which service that port corresponds to, and runs the configured program, which can be an Apache in *inetd* mode. It is worth noting that some of the more advanced features of Apache are not supported in this mode, so it should only be used in very simple cases. Support for this mode may well be removed in future releases of Apache.

Both cases boil down to an Apache with an incoming connection. Remembering our first statement in this section, that the object of the whole exercise is to resolve the incoming request into a filename, Apache first determines which IP address and port number were used by asking the operating system internally. You, the webmaster, do not have to bother about this.

Apache then uses the IP address and port number to decide which virtual host is the target of this request. The virtual host then takes the path, which was handed to it in the request, and reads that against its configuration to decide on the appropriate response—which it returns.

Most of this book is about the possible appropriate responses and how Apache decides which one to use.

Compiling Apache

The current versions of Apache run under UNIX to get the necessary multi-tasking, so this edition of the book is about compiling and running the software under this operating system. We may deal with Windows NT and OS/2 versions in later editions.

People who are new to UNIX might like to look at O'Reilly's *Learning the UNIX Operating System* by Grace Todino, John Strang, and Jerry Peek.

Which Apache?

As we were writing this book, Apache 1.1.1 was the current release, but we wrote about Apache 1.2, which was under development. In this chapter we assume that Apache 1.2 has been released by the time this book appears. If it has not, the fact will be clear enough from the filenames on the Apache server. We suggest that you go for the newest release that is out of beta test (beta test versions have a “b” in their filenames).

Which UNIX?

We experimented with SCO UNIX and QNX (which both support Apache) before settling on FreeBSD as the best environment for this exercise. The whole of FreeBSD is available—free—from <http://www.freebsd.org>, but \$40 to Walnut Creek (at <http://www.cdrom.com>) gets you two CD-ROMS with more software on them than you can shake a stick at (including all the sources) plus a rather Spartan manual that should just about get you going. Without Walnut Creek’s manual, we think it would cost a lot more than \$40 in spiritual self-improvement.

If you use FreeBSD, you will find (we hope) that it installs from the CD-ROM easily enough, but that it lacks several things you will need later. Among these are Perl, Emacs, and some better shell than *sh* (we like *bash* and *ksh*), so it might be sensible to install them straightway from their lurking places on the CD-ROM.

gzip

Most UNIX software available on the Web is compressed using *gzip*, a GNU compression tool. If you haven’t a copy, it would be a good idea to get one. It is easy enough to find from any web browser:

```
ftp://your_mirror_site
```

Or you can access it from a command-line *ftp* package:

```
% ftp your_mirror_site
```

Go to *packages/gnu* (or wherever the site mirrors the GNU software). If you don’t know where your local GNU mirror site is, take a look at <http://www.gnu.org/>.

On arriving at this directory, you should read another *README*. This explains the various versions of *gzip* that are available. Some are in diff files, containing the differences from earlier sources; probably not much use for first-time users. Others end in *.gz*, implying that they themselves are *gzip*’d—equally no good to someone who has not got it already. However, you should find *gzip.1.2.4.tar*, *gzip.1.2.4.tar.Z*, and *gzip.1.2.4.msdos.exe*. The *.tar* version contains the uncom-

pressed sources; the *.tar.Z* version is compressed using the UNIX *compress* utility, and is much smaller. The *.msdos.exe* version is a self-extracting MS-DOS file of the executable to run under MS-DOS. We would advise the *.tar.Z* version. If that fails (possibly because your UNIX's *compress* is incompatible with theirs), then try the *.tar* even though it is large. The MS-DOS *gzip* could well cause trouble because of the different attitudes UNIX and MS-DOS have about line endings and filenames.

Having downloaded the *.tar.Z* file, and if necessary, copied it to your UNIX machine, you can set about building it. Make a directory for it in some suitable spot, copy the file into it, and decompress it with:

```
% uncompress filename
```

or:

```
% compress -d filename
```

Then, unarchive it with:

```
% tar xvf gzip.tar
```

xvf means extract verbosely from file. This creates a subdirectory called *gzip-1.2.4* in which are all the files. So, boldly:

```
% cd gzip-1.2.4
```

Read the *README* as usual. Read *INSTALL*. It tells you to run *./configure*. This checks for a lot of stuff that causes people doing UNIX ports to have a bad day and hopefully leads to a pleasant afternoon. Now run *make* to compile and link all the files. If this seems to run properly, type:

```
% make check
```

This runs an automatic check. Being cynical, we were amazed to find that it said *gzip test OK*. Finally, to get the executable bedded down in your system, type:

```
% make install
```

You might be surprised to see a complete recompile taking place. This is because all the *.o* files are dependent on the *Makefile* itself. Here we had a smidgen of trouble because it wanted to put the executables, manuals, and so forth below */usr/local*. We didn't have this directory, but it is a sensible one to have because it avoids the problem of either accidentally obliterating commands that came with your UNIX, or your UNIX deciding you shouldn't have that utility in that place and obliterating it for you. (Some UNIX systems have security scripts that check for this kind of thing.)

However, if we had wanted to put it all under */usr*, *INSTALL* explained that we could specify a different path by running:

```
% ./configure --prefix=/usr
```

Now, we could type:

```
% make install
```

Finally, you can tidy up and free some space locally by removing binaries and objects. Type:

```
% make clean
```

Making Apache

Load Apache from the enclosed CD-ROM or download it from a suitable mirror site, which can be found at <http://www.apache.org/>. You will get a compressed file, with the extension *.gz* if it has been *gzip*'d, or *.Z* if it has been compressed. When expanded, the Apache *.tar* file creates a tree of subdirectories, and each new release does the same, so you need to create a directory on your FreeBSD machine where all this can live sensibly. Get into your new Apache directory, copy the *apache.tar* file there and uncompress the *.Z* version or *gunzip* the *.gz* version.

Now unpack it using *tar*:*

```
% tar xvf apachename.tar
```

There are a number of files with names in capital letters like *README* that look as if you ought to read them. *README* tells you first of all how to compile Apache, and the first thing it wants you to do is to go to the *src* subdirectory and read *INSTALL*.

The next thing to do is to edit the file *Configuration*. (If you have downloaded a beta test version you first have to copy *Configuration.tmpl* to *Configuration*.)

We have to edit *Configuration* to set things up properly. The whole file is in Appendix A of the installation kit. A script called *Configure* then uses *Configuration* and *Makefile.tmpl* to create your operational *Makefile*. (Don't attack *Makefile* directly; any editing you do will be lost as soon as you run *Configure* again.)

Configuration, as it explains, has four kinds of things in it:

- Comments: lines starting with #
- Rules: start with the word **Rule**
- Commands to be inserted into *Makefile*: lines starting with nothing
- Module selection lines beginning with **Module**

* If you are using GNU *tar*, it is possible to uncompress and untar in one step: `tar zxvf apachename.tar.gz`.

For the moment, we will only be reading the comments and occasionally turning a comment into a command by removing the leading #, or vice versa. Most comments are in front of optional module inclusion lines. These modules are self-contained sections of source code dealing with various functions of Apache that can be compiled in or left out. You can also write your own module if you want. Inclusion of modules is done by uncommenting (removing the leading #) lines in *Configuration*. The only drawback to including more modules is an increase in the size of your binary and an imperceptible degradation in performance.*

Since this book tries to exercise all the standard functions of Apache, we compiled almost all the modules. The lines in *Configuration* we left uncommented were:

```
Module mime_module      mod_mime.o
Module access_module    mod_access.o
Module auth_module      mod_auth.o
Module negotiation_module mod_negotiation.o
Module includes_module  mod_include.o
Module dir_module       mod_dir.o
Module cgi_module       mod_cgi.o
Module userdir_module   mod_userdir.o
Module alias_module     mod_alias.o
Module env_module       mod_env.o
Module config_log_module mod_log_config.o
Module asis_module      mod_asis.o
Module imap_module      mod_imap.o
Module action_module    mod_actions.o
Module browser_module   mod_browser.o
Module agent_log_module  mod_log_agent.o
Module referer_log_module mod_log_referer.o
Module rewrite_module    mod_rewrite.o
Module status_module    mod_status.o
Module info_module      mod_info.o
Module anon_auth_module  mod_auth_anon.o
Module db_auth_module   mod_auth_db.o
Module dbm_auth_module  mod_auth_dbm.o
Module digest_module    mod_digest.o
Module expires_module    mod_expires.o
Module headers_module    mod_headers.o
Module usertrack_module  mod_usertrack.o
Module proxy_module      modules/proxy/lib_proxy.a
```

The three lines we left commented were:

```
# Module cern_meta_module  mod_cern_meta.o
# Module mysql_auth_module  mod_auth_mysql.o
# Module dld_module        mod_dld.o
```

* Assuming the module has been carefully written, it does very little unless enabled in the *httpd.conf* files.

The first, `cern_meta_module`, concerns long-ago backward compatibility with CERN's server. The second, `mysql_auth_module`, concerns SQL database management of a large file of user passwords, using the shareware package Minerva SQL, often known as mSQL, and it is useless unless you have bought it. We had not and do not suppose that you have either, so there was no occasion for it. The third, `dld_module`, concerns some very experimental dynamic-link loading of extra code. It seems that FreeBSD does not support its end of the business, so we ignored it too.

Although we've mentioned `db_auth_module` and `dbm_auth_module` above, they provide equivalent functionality and shouldn't be compiled together.

These are the "standard" Apache modules, approved and supported by the Apache Group as a whole. There are a number of other modules available: see Chapter 12, *Extra Modules*.

Later on, when we are writing Apache configuration scripts, we can make them adapt to the modules we include or exclude with the `IfModule` directive. This allows you to give out predefined Config files that always work (in the sense of Apache loading) whatever mix of modules is actually compiled in. Thus, for instance, we can adapt to the absence of configurable logging with the following:

```
...
<IfModule config_log_module>
LogFormat "customers: host %h, logname %l, user %u, time %t, request
%r, status %s,bytes %b,"
</IfModule> ...
```

The rules in the *Configuration* file allow you to adapt for a few exotic configuration problems. The syntax of a rule in *Configuration* is:

```
Rule RULE=value
```

The possible *values* are:

yes

Configure does what is required;

default

Configure makes a best guess

Any other *value* is ignored.

The rules are:

WANTHSREXEX

Use Apache's regular expression package rather than your operating system's;

SOCKS4

SOCKS is a firewall traversal protocol that requires client-end processing. See <http://ftp.nec.com/pub/security/socks.cstc>. If set to **yes**, be sure to add the

SOCKS library location to `EXTRA_LIBS`, otherwise *Configure* assumes `L/usr/local/lib-lsocks`. This allows Apache to make outgoing SOCKS connections, which is not something it normally needs to do, unless it is configured as a proxy. Although the very latest version of SOCKS is SOCKS5, SOCKS4 clients work fine with it.

STATUS

If **yes**, and *Configure* decides that you are using the Status module, then full status information is enabled. If the status module is not included, **yes** has no effect.

BADMMAP

If Apache compiles all right but refuses to run with the complaint:

```
httpd: could not mmap memory
```

there is a problem sharing memory between processes; you should set this rule to **yes** and report this problem to the Apache Group.

IRIXNIS

If *Configure* decides that you are running SGI IRIX, and you are using NIS, set this to **yes**.

A recent, and useful, improvement to Apache is that it automatically detects your operating system and adjusts the *Configuration* file to suit. If it goes wrong you will have to edit *src/Configuration*, *src/helpers/GuessOS*, and *src/conf.h* to solve the problems. A file with the latest information about porting Apache is maintained in *src/PORTING*.

The *INSTALL* file (Appendix B of the installation kit) says that all we have to do now is run the configuration script by typing `% Configure`. If you type this literally you get the message `%: not found`. What they mean is that when you see `%` (the UNIX prompt), you should type *Configure*:

```
% Configure
```

If you have logged on as *root*, or possibly even if you haven't, you may need to type:

```
% ./Configure
```

You should see:

```
Using Configuration as config file
Configured for FreeBSD/NetBSD platform
```

or whatever your operating system is. Then type:

```
% make
```

* This is because the current directory is left off the executable search path for security reasons.

When you run *make*, the compiler is set going, and streams of reassuring messages appear on the screen. However, things may go wrong that you have to fix: this can appear more alarming than it really is. For instance, in an earlier attempt to install Apache on a SCO machine, we got the compile error:

```
Cannot open include file 'sys/socket.h'
```

Clearly (since sockets are very TCP/IPish things), this had to do with TCP/IP, which we had not installed: we did so. Not that this is any big deal, but it illustrates the sort of minor problem that arises. Not everything turns up where it ought to. If you find something that really is not working properly—unlike this example—it is sensible to post a message to the *bugs* mailing list (*apache-bugs@apache.org*) to report the problem in case it has wider implications.

The result of *make* was the executable *httpd*. If you run it with:

```
% ./httpd
```

It complains that it:

```
could not open document config file /usr/local/etc/httpd/conf/
httpd.conf
```

This is not surprising because it doesn't exist. Before we are finished, we will become very familiar with this file. It is perhaps unfortunate that it has a name so similar to the *Configuration* file we have been dealing with here, because it is quite different. We hope that the difference will become apparent later on.

Installing Apache

Once the excitement of getting Apache to compile and run died down, we reorganized things in accordance with the system defaults.

We copied the executable *httpd* to the directory */usr/local/bin* to put it on the path. You can automate this by editing *Makefile.tmpl*. Find the lines:

```
httpd: $(OBSJ)
$(CC) $(LFLAGS) -o httpd ...
```

Add, immediately below:

```
<TAB>cp httpd /usr/local/bin
```

Note that this line *must* start with a Tab, because that is how *make* identifies the commands it carries out. Now, each time you run *Configure*, the new *Makefile* will have this line in it. This evaporates next time you download a new version of Apache into this location, because the download brings a new *Makefile.tmpl* with it.

2

Our First Web Site

In this chapter:

- *WebUser and Webgroup*
- *httpd's Flags*
- *Permissions*
- *A Local Network*
- *An Experimental Web Site*
- *site.simple*

We now have a shiny bright *httpd*, ready for anything. As we shall see below, we will be creating a number of demonstration web sites. For the first, we made a subdirectory */usr/www/site.toddle*. Since you may want to keep your demonstration sites somewhere other than */usr/www*, we normally refer to this path as *.../*. So we will talk about *.../www/site.toddle*, or *.../site.toddle*.

In *.../site.toddle*, we created the three subdirectories Apache expects: *conf*, *logs*, and *htdocs*. The *README* in Apache's root directory says:

The next step is to edit the configuration files for the server. In the subdirectory called *conf* you should find distribution versions of the three configuration files: *srn.conf-dist*, *access.conf-dist*, and *httpd.conf-dist*. Copy them to *srn.conf*, *access.conf*, *httpd.conf* respectively.

First edit *httpd.conf*. This sets up general attributes about the server—the port number, the user it runs as, etc. Next edit the *srn.conf* file—this sets up the root of the document tree, special functions like server-parsed HTML or internal imagemap parsing, etc. Finally, edit the *access.conf* file to at least set the base cases of access. Documentation for all of these is located at *http://www.apache.org/docs/*. Finally, make a call to *httpd*, with a *-f* to the full path to the *httpd.conf* file.

And voila! The server should be running.

This is excellent advice if we knew how to edit these three *.conf* files, but at the moment we do not. We will learn more later. A simple expedient for now is to run Apache with no configuration and let it prompt us for what it needs.

We can point *httpd* at our site with the *-d* flag (notice the full pathname to the *site.toddle* directory):

```
% httpd -d /usr/www/site.toddle
```

Since you will be typing this a lot, it's sensible to copy it into a script called *go* in *site.toddle* by typing:

```
% cat > go
httpd -d /usr/www/site.toddle
^D
```

^D is shorthand for *Ctrl-D*, which ends the input and gets your prompt back. Make *go* runnable and run it by typing:

```
% chmod +x go
% ./go
```

(Note *./go* rather than *go*: we write this because we logged on as root on our FreeBSD machine, and UNIX makes this syntax necessary as a security measure. If you do not log on as root, *go* may do, but *./go* does no harm.)

This launches Apache in the background. Check that it's running by typing:

```
% ps -aux
```

This UNIX utility lists all the processes running, among which you should find *httpd* (*-a* gives information about other users, *-x* about processes detached from their terminals).^{*} The advantage to this way of running Apache is that you can now start up a browser on the same console and pretend to be a client. This makes the initial testing very easy.

Sooner or later, you are finished testing and want to stop Apache. In order to do this you have to get its process identity (PID) using *ps -aux* and execute the UNIX utility *kill*:

```
% kill PID
```

Alternatively, since Apache writes its PID in the file *.../logs/httpd.pid*, you can write yourself a little script:

```
kill 'cat /usr/www/site.toddle/logs/httpd.pid'
```

After a while this gets tedious. The alternative is to add the flag *-X* to the command line for *httpd* in *go*. This makes Apache run in the foreground so it can be stopped by *^C* or *DEL*, depending on your flavor of UNIX. Since it runs in the foreground, it monopolizes a console, but you can go to another console (with *ALT Fn*, on many systems, or by starting another *telnet* session) to test it with a browser. This is probably the best way to test Apache, but, please note that this mode is bad for a production server. Only one copy of Apache is running, instead of the usual "gang," so it will be slow if even mildly loaded. It can also cause mysterious pauses in your browser. The *-X* flag is for debugging only.

^{*} On System V-based UNIX systems (as opposed to Berkeley-based), the command *ps -ef* should have a similar effect.

Whichever you do, nothing appears to happen, but when we look in the *logs* subdirectory, we find a file called *error_log* and in it the entry:

```
[<date>] - couldn't determine user name from uid
```

This means more than might at first appear. We had logged in as *root*. Because of the security worries of letting outsiders log in with superuser powers, Apache, having been started with root permissions so that it can bind to port 80, has changed its userid to -1. On many UNIX systems this corresponds to *nobody* in *nogroup*: a harmless person. However, it seems that FreeBSD does not understand this notion, hence the error message.*

WebUser and Webgroup

The remedy is to create a new person, who logs in safely without a shell, called *webuser*. He belongs to *webgroup*. On a FreeBSD system, you can run *adduser* to make this new person *webuser*.

```
Enter username [a-z0-9]: webuser
Enter full name[]: webuser
Enter shell bash csh date no sh tcsh [csh]: no
Uid [some number]:
Login group webuser [webuser]: webgroup
Login group is 'webgroup'.q. Invite webuser into other
groups: guest no [no]:
Enter password []: password
```

You then get the report:

```
Name:webuser
Password: password
Fullname: webuser
Uid: some number
Groups:webgroup
HOME:/home/webuser
shell/nonexistent
OK? (y/n) [y]:

send message to 'webuser'.xd5 and: no route second_mail_address [no]:
Add anything to default message (y/n) [n]:
Send message (y/n) [y]: n
Add another user? (y/n) [y]:n
```

The bits of the report after OK are really irrelevant, but of course FreeBSD does not know that you are making a nonexistent user. Having told the operating system about this user, you now have to tell Apache. Edit the file *httpd.conf* to include the lines:

* In fact, this problem was fixed for FreeBSD shortly before going to press, but you may still encounter it on other operating systems.

```
User webuser
Group webgroup
```

When you run Apache now, you get the error message:

```
httpd: cannot determine local hostname
Use ServerName to set it manually.
```

What Apache means is that you should put a line in the *httpd.conf* file:

```
ServerName localhost
```

Finally, before you can expect any action, you need to set up some documents to serve. Apache's default document directory is *.../httpd/htdocs*—which you can't use because you are at */usr/www/site.toddle*—so you have to set it explicitly. Create *.../site.toddle/htdocs*, and then in it a file called *1.txt* containing the immortal words: *hullo world*. Then add a line to *httpd.conf*:

```
DocumentRoot /usr/www/site.toddle/htdocs
```

Now, when you fire up *httpd*, you should have a working web server. To prove it, move to another console (or another machine) and start up a browser to access our new server. If you are on the same machine, aim the browser at *http://127.0.0.1/*; if on another machine, point to *http://yourmachinename/*.

http://127.0.0.1/ might look a bit odd as a web address. However, this is the “real” IP address—the “loopback” IP address of your own machine—and it is standard wherever TCP/IP is found. If you don't believe this, look in */etc/hosts* and there it is. Since it maps to *localhost* in that file, use that: *http://localhost/*.

As we know, *http* means use the HTTP protocol to get documents, and */* on the end means go to the *DocumentRoot* directory you set in *httpd.conf*. Lynx is the text browser that comes with FreeBSD; if you are using this operating system, type:

```
% lynx http://127.0.0.1/
```

or:

```
% lynx http://localhost/
```

You see:

```
INDEX OF /
* Parent Directory
* 1.txt
```

If you move to *1.txt* with the down arrow, you see:

```
hullo world
```

If you don't have Lynx on your server, you can use *telnet*:

```
% telnet localhost 80
```

Then type:

```
GET / HTTP/1.0 <CR><CR>
```

You should see:

```
HTTP/1.0 200 OK
Sat, 24 Aug 1996 23:49:02 GMT
Server: Apache/1.2-dev
Connection: close
Content-Type: text/html

<HEAD><TITLE>Index of </TITLE></HEAD><BODY>
<H1>Index of </H1>
<UL><LI> <A HREF="/"> Parent Directory</A>
<LI> <A HREF="1.txt"> 1.txt</A>
</UL></BODY>
Connection closed by foreign host.
```

The stuff between the < and > is HTML, written by Apache, which, if viewed through a browser, produces the formatted message shown by Lynx above, or by Netscape in the next chapter.

As an educational exercise, run:

```
% ps -aux
```

to get a display of all the processes running. Among a lot of UNIX stuff, you will find one copy of `httpd`. If you stop Apache with `^C`, remove the `-X` flag from `go`, restart it, and try again, you will see six copies of `httpd`. The first one is owned by `root` and is the one you started; the next five are almost exact copies, preconfigured and ready to run. They have been forked by the first copy, are owned by `webuser` (who, you'll remember, is safely unprovided with a shell, so nothing much bad can happen if he/she/it is hijacked), and sit waiting for a call to come in on port 80.

The `root` copy is still attached to port 80—so that its children will be—but is not listening. This is because it is `root` and has too many powers. It is necessary for this “master” copy to remain running as `root` as only `root` can open ports below 1024. Its job is to monitor the scoreboard where the other copies post their status: busy or waiting. If there are too few waiting (default 5, set by the `MinSpareServers` directive in `httpd.conf`), the `root` copy starts new ones; if there are too many waiting (default 10, set by the `MaxSpareServers` directive), it kills some off. If you note the PID (shown by `ps -ax`, also to be found in `.../logs/httpd.pid`) of the `root` copy and kill it, with:

```
% kill PID
```

you will find that the other five disappear as well; this is the nature of forked programs. If you took `-X` out of `go`, now would be a good time to put it back.

httpd's Flags

httpd takes the following flags:

- d Specifies an alternate initial `ServerRoot`
- f Specifies an alternate `ServerConfig` file
- v Shows version number
- h Lists directives
- X Runs a single copy (only for debugging)

The Apache Group seems to put in extra flags quite often, so it is worth experimenting with `httpd -?` to see what you get.

Permissions

If Apache is to work properly, it's important to correctly set the file-access permissions. In UNIX systems, there are three kinds of permission: read, write, and execute, and they attach to each object in three levels: user, group, and other or rest of world. If you have installed the demonstration sites, go to `.../site.cgi/htdocs` and type:

```
% ls -l
```

You see:

```
-rw-rw-r-- 5 root bin 1575 Aug 15 07:45 form_summer.html
```

The first “-” indicates that this is a regular file, and it is followed by three permission fields, each of three characters. They mean, in this case:

- User (`root`): read yes, write yes, execute no
- Group (`bin`): read yes, write yes, execute no
- Other: read yes, write no, execute no

When the permissions apply to a directory, the “x”—execute permission—means “scan,” the ability to see the contents and move down a level.

The permission that interests us is *other* because the copy of Apache that tries to access this file belongs to user *webuser* and group *webgroup*. These were set up to have no affinities with *root* and *bin*, so it can gain access only under the *other* permissions, and the only one set is “read.”

Consequently, a Bad Guy who crawls under the cloak of Apache cannot alter or delete our precious *form_summer.html*; he can only read it.

We can now write a coherent doctrine on permissions. The way we have set things up is that everything in our web site has owner *root* and group *wheel*, except the data vulnerable to attack.

We did this partly because it is a valid approach, but also because it is the only portable one. The files on our CD-ROM with owner *root* and group *wheel* have owner and group numbers "0" that translate into similar superuser access on every machine.

Of course, this only makes sense if the webmaster has root login permission, which we had. You may have to adapt the whole scheme if you do not have root login, and you should perhaps consult your site administrator.

In general, on a web site, everything should be owned by a user who is not *webuser* and a group that is not *webgroup* (assuming you use these terms for Apache configurations).

There are four kinds of files that we want to give *webuser* access to:

- directories
- data
- programs
- shell scripts

webuser must have scan permissions on all the directories starting at root down to wherever the accessible files are. If Apache is to access a directory, that directory and all in the path must have *x* permission set for *other*. You do this by entering:

```
% chmod o+x each-directory-in-the-path
```

In order to produce a directory listing (if this is required by, say, an index), the final directory must have read permission for *other*. You do this by typing:

```
% chmod o+r final-directory
```

It probably should not have write permission set for *other*:

```
% chmod o-w final-directory
```

In order to serve a file as data—and this includes, for example, files like *.btaccess* (see Chapter 3, *Towards a Real Web Site*)—the file must have read permission for *other*:

```
% chmod o+r file
```

And, as before, deny write permission:

```
% chmod o-w file
```

In order to run a program, the file must have execute permission set for *other*:

```
% chmod o+x program
```

In order to execute a shell script, the file must have read and execute permission set for *other*:

```
% chmod o+rx script
```

A Local Network

Emboldened by the success of *site.toddle*, we can now set about a more realistic setup, without as yet venturing out onto the unknown waters of the Web.

We need to get two machines, one running Apache under some sort of UNIX, linked to another running a GUI browser.

There are two main ways this can be achieved:

- Run Apache and a browser (such as Mosaic or Netscape under X) on the same UNIX machine. The “network” is then provided by UNIX.
- Run Apache on a UNIX box and a browser on a Windows 95/Windows NT/MacOS machine and link them with Ethernet (which is what we did for this book using FreeBSD).

We cannot hope to give detailed explanations for all these situations. We expect that many of our readers will already be webmasters, familiar with these issues, who will want to skip the next section. Those who are new to the Web may find it useful to know what we did.

An Experimental Web Site

First, we had to install a network card on the FreeBSD machine. As it boots up, it tests all its components and prints a list on the console, which includes the card and the name of the appropriate driver. We used a 3Com card, and the following entries appeared:

```
...
1 3C5x9 board(s) on ISA found at 0x300
ep0 at 0x300-0x30f irq 10 on isa
ep0: aui/bnc/utp[*BNC*] address 00:a0:24:4b:48:23 irq 10
...
```

This indicated pretty clearly that the driver was *ep0*, and that it had installed properly. If you miss this at boot up, you can hit the **Scroll Lock** key and hit **Page Up** till you see it, then hit **Scroll Lock** again to return to normal operation.

Having got a card working, we needed to configure its driver *ep0*. We did this with the script *lan_setup*:

```
ifconfig ep0 192.168.123.2
ifconfig ep0 192.168.123.3 alias netmask 0xFFFFFFFF
ifconfig ep0 192.168.124.1 alias
```

The *alias* command makes *ifconfig* bind an additional IP address to the same device. The *netmask* command is needed to stop FreeBSD from printing an error message (for more on netmasks, see O'Reilly's *TCP/IP Network Administration*).

Note that the network numbers used here are suited to our particular network configuration. You'll need to talk to your network administrator to determine suitable numbers for your configuration. Each time we start up the FreeBSD machine to play with Apache, we have to run this script from the directory */usr/www* with:

```
% ./lan_setup
```

You might like to ask your network administrator to arrange for this script, or something equivalent, to be run at startup, perhaps by putting it in */etc/rc*.

If you are following the FreeBSD installation or something like it, you also need to install IP addresses and their hostnames (if we are to be pedantic we would call them fully qualified domain names (FQDN) in the file */etc/hosts*:

```
192.168.123.2 www.butterthlies.com
192.168.123.3 sales.butterthlies.com
192.168.124.1 www.faraway.com
```

Note that this only works if you are not using DNS on your machine. Since the names we use here are invented for testing and not registered with the InterNIC, we cannot easily use DNS to support the names.

Our other machine was running Windows 95. We installed a network card there in the usual way and via **Control Panel/Network** made sure that the card was bound to TCP/IP and that the TCP/IP interface had an address on the same network as the Apache IP address: 192.168.123.1.

Type *./lan_setup* on the UNIX machine to configure the network interface. Test that it is working by sending data to the other machine using *ping*. Type:

```
% ping 192.168.123.1
```

You should see a series of reports of packets sent. If *ping* reports *Host is down*, the network isn't working.

Since we were not going to waste Internet bandwidth on our fumbblings, we had to fool Netscape into thinking that it knew the DNS registrations for the IP addresses we set up in *lan_setup*. The simple way to do this is just to edit the file *\windows\hosts* on the client machine if it happens to be a Windows machine

running Microsoft TCP/IP (most TCP/IP implementations have a similar facility). On the UNIX machine, the file is */etc/hosts*. You should find the entry:

```
127.0.0.1 localhost
```

After it, include the following:

```
192.168.123.2 www.butterthlies.com
192.168.123.3 sales.butterthlies.com
```

Why these names? Read on!

In order to carry out these exercises, you should stop Netscape logging on to the Internet. In our case we did this simply by not running the separate dial-up program. The first time in each session that you ask Netscape to log on to, say, *www.butterthlies.com*, a long pause will ensue while Netscape ponders this sad and unusual situation, tries to contact a DNS server to get an IP address, gives up, and grumpily looks in its own navel—as it were. After that it bites the bullet and is more brisk. You can speed things up by using the IP address instead of the hostname.

site.simple

site.simple is *site.toddle* with a few small changes. The script *go* is different in that it refers to *.../site.simple/conf/httpd.conf* rather than *.../site.toddle/conf/httpd.conf*.

```
% httpd -d /usr/www/site.simple
```

This will be true of each site in the demonstration setup, so we will not mention it again.

The most important change is that we will be ignoring the instructions that come with the Apache sources to make three configuration files (*httpd.conf*, *srm.conf*, and *access.conf*). In fact it is not necessary to have three files. In our view, it is better to have a single file because you can see at a glance how things are arranged. And, in one case we discovered, it is actually very dangerous to spread configuration around (see Chapter 5, *Authentication*).

Include the following lines in *httpd.conf*:

```
AccessConfig /dev/null
ResourceConfig /dev/null
```

Now, simply delete or fail to create *srm.conf* and *access.conf*. The effect of these settings is to switch off the use of these files.

Now you can put all your configuration directives in *httpd.conf*.

You will find that this is what we have done throughout this book. Furthermore, we refer to this file as the “Config” file to distinguish it from the *Configuration* file used at compile time—see Chapter 1, *Getting Started*.

The `.../conf/httpd.conf` file now contains:

```
User webuser
Group webgroup
ServerName localhost
DocumentRoot /usr/www/site.simple/htdocs
AccessConfig /dev/null
ResourceConfig /dev/null
```

Note the simplifying lines:

```
AccessConfig /dev/null
ResourceConfig /dev/null
```

In `.../htdocs` we have, as before, `1.txt`:

```
hullo world!
```

Now, type `./go` on the server. Go to the client machine and retrieve `http://www.butterbflies.com`. You should see:

```
Index of /
. Parent Directory
. 1.txt
```

Click on `1.txt` for an inspirational message.

This all seems satisfactory, but there is a hidden mystery. We get the same result if we connect to `http://sales.butterbflies.com`. Why is this? Why, since we have not mentioned either of these URLs or their IP addresses in the configuration file on *simple.site* do we get any response at all?

The answer is that when we ran `lan_setup` on the server, we told the device `ep0` to respond to any of the IP addresses:

```
192.168.123.2
192.168.123.3
192.168.124.1
```

By default Apache listens to all IP addresses belonging to the machine and responds alike to them. If there are virtual hosts configured, Apache runs through them, looking for an IP name that corresponds to the incoming connection; Apache uses that configuration if it is found, or the main configuration if it is not. In later chapters we look at more definite control with the directives: `BindAddress`, `Listen`, and `<VirtualHost...>`.

It has to be said that working like this seemed to get Netscape into a rare muddle and that to be sure the server was working properly it was often necessary to clear Netscape's cache files in between re-logons by going to `Options/Network`

Preferences/Clear Memory Cache Now and Clear Disk Cache Now. If you don't, Netscape tends to make itself a jumble of several different responses from the server. In extreme cases, we had to exit from Netscape and reload it between tests to make sure that it was accessing the site from a standing start. This is because we are doing what no administrator would normally do, flipping around between different versions of the same site with different versions of the same file. Whenever we flip from a newer version to an older version, Netscape is led to believe that its cached version is up-to-date.

Back on the server, stop Apache with `^C` (or whatever your kill character is) and look at the log files. In `.../logs/access_log` you should see something like:

```
192.168.123.1 -- [<date-time>] "GET / HTTP/1.0" 200 177
```

200 is the response code (meaning, "OK, cool, fine"), and 177 is the number of bytes transferred. In `.../logs/error_log`, there should be nothing because nothing went wrong. However, it is a good habit to look there from time to time, though you have to make sure that the date and time logged correspond to the problem you are investigating. It is easy to fool yourself with some long-gone drama.

In `.../logs/httpd.pid` we find the PID of the copy of Apache that was running. If we had not used the `-X` flag in `go`, this is the number we would use with `kill` to stop Apache:

```
% kill PID
```

Failing this file, we would use `ps -ax` to find the PID.

```
# Translate the client X509 into a Basic authorisation. This means that
the
# standard Auth/DBMAuth methods can be used for access control. The user
name
# is the "one line" version of the client's X509 certificate. Note that no
# password is obtained from the user. Every entry in the user file needs
this
# password: xxj3lZMTZzkVA. See the code for further explanation.
SSLFakeBasicAuth

# A home for miscellaneous rubbish generated by SSL. Much of it is
duplicated
# in the error log file.
SSLLogFile /tmp/ssl.log

# New and undocumented directives
#SSLRequiredCiphers
#SSLRequireCipher
#SSLBanCipher

# Experiment with authorization...
#<Directory /u/ben/www/1/docs>
#AuthType Basic
#AuthName Experimental
#AuthGroupFile /dev/null
#AuthUserFile /u/ben/www/1/users
#<Limit PUT GET>
#allow from all
#require valid-user
#</Limit>
#</Directory>
```

The port issue is simply that if you are the superuser (you logged in as root), then ports below 1024 are accessible to you and the programs you run, and you can use the SSL default, 443. If not, 8887 would be a reasonable choice. If you use 443, your clients simply have to log on to <http://www.butterflies.com/>; if 8887, they have to mention the port and log on to <http://www.butterflies.com:8887/>.

Remember to edit `go` so it invokes `httpsd` (the secure version); otherwise, Apache will rather puzzlingly object to all the nice new SSL directives. Run `./go` in the usual way and log on. Netscape displays an unbroken key in the bottom left corner and a handsome blue stripe along the top. Evidently Netscape's product liability team have been to work here, and you are taken through an amazing rigmarole of legal safeguards and "are you absolutely sure?"s.

We were running with `SSLVerifyClient 0`, so Apache made no enquiry into our credibility as a client. Change it to 2, to force the client to present a valid certificate. Netscape says:

```
the site 192.163.123.2 has requested client authentication, but you do
not have a Personal Certificate to authenticate yourself.
```

Oh, the shame of it.

The simple way to fix this smirch is to get a beta certificate from <http://thawte.com/>, or any other certificate authority recognized by your browser. Log onto their site, and follow the instructions. It will install itself into your Netscape v3.

But this is only half the story. If we are going to be strict at the Apache end, we want to make sure that Thawte are who they say they are and we therefore want their own certificate. Unhappily, Thawte deleted their root certificate from the web site before we could download it, so we failed to make all this work. You have to understand that the whole web security business is still in its infancy.

If you log onto <http://www.butterthlies.com> again, you will be asked if you want to submit the new Thawte certificate.

Apache-SSL's Directives

Apache-SSL's directives follow:

SSLDisable

`SSLDisable`

Disable SSL. This is useful if you wish to run both secure and nonsecure hosts on the same server.

SSLCACertificatePath

`SSLCACertificatePath` *directory*

This is the path to the directory where you keep the public keys of the certification authorities whose clients you deal with. They must be PEM-encoded.

SSLCACertificateFile

`SSLCACertificateFile` *filename*

This is your certificate verification file, which must be PEM-encoded.

SSLCertificateFile

`SSLCertificateFile` *filename*

This is your PEM-encoded certificate. It is encoded with distinguished encoding rules (DER), and ASCII armored so it will go over the Web. If the certificate is encrypted, you are prompted for a pass phrase.

SSLCertificateKeyFile

`SSLCertificateKeyFile filename`

This is the private key of your certificate. If the key is not combined with the certificate, use this directive to point at the key file. If the filename starts with /, it specifies an absolute path; otherwise, it is relative to the default certificate area that is currently defined by `SSLLeay` to be either: `/usr/local/ssl/private` or *<wherever you told ssl to install>/private*.

SSLVerifyClient

`SSLVerifyClient number`
default: 0

This directive defines what you require of clients:

- 0: no certificate required.
- 1: the client *may* present a valid certificate.
- 2: the client *must* present a valid certificate.
- 3: the client may present a valid certificate, but not necessarily from a certification authority for which we hold a key.

SSLVerifyDepth

`SSLVerifyDepth number`

In real life, the certificate we are dealing with was issued by a CA who in turn relied on another CA to validate them, and so on, back to a root certificate. This directive specifies how far up or down the chain we are prepared to go before giving up. What happens when we give up is determined by the setting given to `SSLVerifyClient`.

SSLFakeBasicAuth

`SSLFakeBasicAuth`

This makes Apache pretend that the user has been logged in using basic authentication (see Chapter 5, *Authentication*), except instead of the username you get the one-line X509, a version of the client's certificate. If you switch this on, with `SSLVerifyClient`, you should see the results in one of the logs. The code adds a predefined password. It's probably better to use `htpasswd` to generate the user file, because *crypt* seems to vary from system to system.

SSLLogFile

SSLLogFile filename

This specifies where SSL puts its logs. Much of the material also goes into `../logs/error_log`.

Cipher Suites

The SSL protocol does not restrict clients and servers to a single encryption brew for the secure exchange of information. There are a number of possible cryptographic ingredients, but as in any cookpot, some ingredients go better together than others. The seriously interested can refer to Bruce Schneier's *Applied Cryptography*, published by John Wiley & Sons, in conjunction with the SSL specification (from <http://www.netscape.com/>). The list of cipher suites is in the SSLeay software at `../ssl/ssl.b`. The macro names give a better idea of what is meant than the text strings:

```
#define SSL_TXT_NULL_WITH_MD5           "NULL-MD5"
#define SSL_TXT_RC4_128_WITH_MD5        "RC4-MD5"
#define SSL_TXT_RC4_128_EXPORT40_WITH_MD5 "EXP-RC4-MD5"
#define SSL_TXT_RC2_128_CBC_WITH_MD5    "RC2-CBC-MD5"
#define SSL_TXT_RC2_128_CBC_EXPORT40_WITH_MD5 "EXP-RC2-CBC-MD5"
#define SSL_TXT_IDEA_128_CBC_WITH_MD5    "IDEA-CBC-MD5"
#define SSL_TXT_DES_64_CBC_WITH_MD5      "DES-CBC-MD5"
#define SSL_TXT_DES_64_CBC_WITH_SHA      "DES-CBC-SHA"
#define SSL_TXT_DES_192_EDE3_CBC_WITH_MD5 "DES-CBC3-MD5"
#define SSL_TXT_DES_192_EDE3_CBC_WITH_SHA "DES-CBC3-SHA"
#define SSL_TXT_DES_64_CFB64_WITH_MD5_1 "DES-CFB-M1"
#define SSL_TXT_NULL                     "NULL"
```

For most purposes, the webmaster does not have to bother with all this.

SSLRequiredCiphers

SSLRequiredCiphers <cipher list>

This directive specifies a colon-separated list of cipher suites, the text strings above, used by SSLeay to limit what the other end can do. This is a per-server option.

SSLRequireCipher

SSLRequireCipher <cipher list>

This directive specifies a space-separated list of cipher suites, used after the connection is established to verify the cipher. This is a per-directory option.

SSLBanCipher

`SSLBanCipher <cipher list>`

This directive specifies a space-separated list of cipher suites, as per `SSLRequireCipher`, except it bans them. It goes like this: if banned, reject; if required, accept; if no required ciphers listed, accept.

A

Support Organizations

The following organizations provide consultation and/or technical support for the Apache web server:

A.B. Enterprises (FutureFX)

Services: Publishing services, web hosting and design, and custom Intranet/
Internet servers

Contact: Jason S. Clary

Address: 4401 Blystone Lane, Plano, TX 75093

Phone: (972) 596-1196 or (800) 600-0786 (toll free in United States)

Fax: (972) 596-3837

Email: abent@futurefx.com

Web site: <http://www.futurefx.com/>

C2Net Software, Inc.

Services: Produces/sells commercial version of Apache, called Stronghold

Contact: Stronghold Sales 510-986-8770

Address: 1212 Broadway Suite 1400, Oakland, CA 94612

Phone: (510) 986 8770

Email: stronghold-sales@c2.net

Web site: <http://www.c2.net/>

Steam Tunnel Operations

Services: Apache support and development

Web site: <http://www.steam.com/>

UK Web

Services: Technical support and consultancy for Apache. Distributor of Stronghold secure server and SafePassage secure client. *Apache Week* web site for Apache news and technical information.

Contact: Mark Cox, Technical Director

Address: 46 The Calls, Leeds, LS2 7EY, United Kingdom

Phone: +44 (113) 222 0046

Fax: +44 (113) 244 8102

Email: business@ukweb.com

Web sites: <http://www.ukweb.com/>, <http://stronghold.ukweb.com/>,
<http://www.apacheweek.com/>

Zyzzzyva Enterprises

Services: Internet commerce development, technical project management and support, Intranet security, and resource development

Address: P.O. Box 30898, Lincoln, NE 68503-0898

Phone: (402) 438 1848

Fax: (402) 438 1869

Email: info@zyzzzyva.com

Web site: <http://www.zyzzzyva.com/>

B

Echo Program

This is the helper program *echo2.c*

```
#include <stdio.h>
#define CR 13
#define LF 10

void getword(char *word, char *line, char stop) {
    int x = 0,y;

    for(x=0;((line[x]) && (line[x] != stop));x++)
        word[x] = line[x];

    word[x] = '\0'.
    if(line[x]) ++x;
    y=0;

    while(line[y++] = line[x++]);
}

char *makeword(char *line, char stop) {
    int x = 0,y;
    char *word = (char *) malloc(sizeof(char) * (strlen(line) + 1));

    for(x=0;((line[x]) && (line[x] != stop));x++)
        word[x] = line[x];

    word[x] = '\0'.
    if(line[x]) ++x;
    y=0;

    while(line[y++] = line[x++]);
    return word;
}
```

```

char *fmakeword(FILE *f, char stop, int *cl) {
    int wsize;
    char *word;
    int ll;

    wsize = 102400;
    ll=0;
    word = (char *) malloc(sizeof(char) * (wsize + 1));

    while(1) {
        word[ll] = (char)fgetc(f);
        if(ll==wsize) {
            word[ll+1] = '\0'.
            wsize+=102400;
            word = (char *)realloc(word,sizeof(char)*(wsize+1));
        }
        --(*cl);
        if((word[ll] == stop) || (feof(f)) || (!(*cl))) {
            if(word[ll] != stop) ll++;
            word[ll] = '\0'.
            return word;
        }
        ++ll;
    }
}

char x2c(char *what) {
    register char digit;

    digit = (what[0] >= 'A' ? ((what[0] & 0xdf) - 'A'.+10 :
        (what[0] - '0'.);
    digit *= 16;
    digit += (what[1] >= 'A' ? ((what[1] & 0xdf) - 'A'.+10 :
        (what[1] - '0'.);
    return(digit);
}

void unescape_url(char *url) {
    register int x,y;

    for(x=0,y=0;url[y];++x,++y) {
        if((url[x] = url[y]) == '%'. {
            url[x] = x2c(&url[y+1]);
            y+=2;
        }
    }
    url[x] = '\0'.
}

void plustospace(char *str) {
    register int x;

    for(x=0;str[x];x++) if(str[x] == '+'. str[x] = ' ');
}

```

```

int rind(char *s, char c) {
    register int x;
    for(x=strlen(s) - 1; x != -1; x--)
        if(s[x] == c) return x;
    return -1;
}

int getline(char *s, int n, FILE *f) {
    register int i=0;

    while(1) {
        s[i] = (char)fgetc(f);

        if(s[i] == CR)
            s[i] = fgetc(f);

        if((s[i] == 0x4) || (s[i] == LF) || (i == (n-1))) {
            s[i] = '\0';
            return (feof(f) ? 1 : 0);
        }
        ++i;
    }
}

void send_fd(FILE *f, FILE *fd)
{
    int num_chars=0;
    char c;

    while (1) {
        c = fgetc(f);
        if(feof(f))
            return;
        fputc(c, fd);
    }
}

int ind(char *s, char c) {
    register int x;

    for(x=0; s[x]; x++)
        if(s[x] == c) return x;

    return -1;
}

void escape_shell_cmd(char *cmd) {
    register int x,y,l;

    l=strlen(cmd);
    for(x=0; cmd[x]; x++) {
        if(ind("&'.q\"|*?~<>^()[]{}$\\", cmd[x]) != -1) {
            for(y=l+1; y>x; y--)
                cmd[y] = cmd[y-1];

```

```
        l++; /* length has been increased */
        cmd[x] = '\\'.
        x++; /* skip the character */
    }
}
```

C

NCSA and Apache Compatibility

This email was sent by Alexei Kosut to the members of the Apache Group to explain the compatibility problems between the NCSA server and Apache 1.1.1.

There has been some discussion lately about the end of NCSA *httpd* development, and Apache replacing it for once and all, and so forth and so on...anyhow, I just thought I'd take this opportunity to point out what NCSA *httpd* 1.5.2 does that Apache does not currently do, feature and config-file wise:

- NCSA supplements the `Redirect` directive with the `RedirectTemp` and `RedirectPermanent` directives, to allow for 301 redirects as well as 302. This is very simple to do.
- NCSA optionally supports Kerberos authentication. I know there's a module out there that does as well; is it compatible with the NCSA syntax?
- Speaking of auth syntax, NCSA's dbm implementation is different than ours. Namely, where we use:

```
AuthUserFile /some/flat/file  
AuthDBMUserFile /some/dbm/file
```

NCSA uses:

```
AuthUserFile /some/flat/file standard  
AuthUserFile /some/dbm/file dbm
```

(the "standard" is optional). This also applies to `AuthGroupFile` and `AuthDigestFile`. Unfortunately, this isn't really possible with the current Apache config-file handling. I wonder if maybe we shouldn't extend the config-file handling routines to allow more than one module to have the same directive (with the same mask and arg list, hopefully), and allow them to "decline" to handle it, as handlers work. This shouldn't be that hard. I'd look into it.

- Satisfy. There are enough patches floating around; can't we just commit one already? (one that works, hopefully)

- The `KeepAlive` syntax in NCSA *httpd* is different from ours. `KeepAliveTimeout` is the same in both, but we use `KeepAlive` where they use `MaxKeepAliveRequests` (and 0 means different things in the two), and they have an additional `KeepAlive On/Off` directive. It can be made to work, it just doesn't now.
- NCSA supports CERN imagemap format as well as NCSA. Do we? (I forget. We should.)
- NCSA supports SSI-parsed CGI output optionally. I don't think we should do this, at least not until 2.0 (ssi could be rewritten as a filter of sorts, implemented with a stacked discipline or some such).
- You can use "referer allow|deny" in access control sections to deny or allow requests based on the Referer header. This is what `mod_block.c` (in */dist/contrib/modules*) does, but with vastly different syntax.
- Redirect doesn't require a full URL: if you omit the server name, it will redirect to the local server.
- "Redirects in *.htaccess* files can now take regular expressions." I have no idea what this means, but that's what it says in the release notes. I can find no evidence of anything regular-expression-like in the code.
- Built-in FastCGI support. This would be trivial; just grab *mod_fastcgi* and add it to the distribution (they even include a *mod_fastcgi.html* in just the right format to add to our docs. Nice of 'em). Their license even lets us do it without asking them first (though it would probably be polite to). This might be a good idea (or not; the thing's 97k, even larger than *mod_rewrite* and *mod_proxy*), FastCGI seems pretty nice and well-designed (even if half of their web site is an ad for their web server). Does anyone have any experience with it?

I think that's about it.

D

SSL Protocol

This appendix reproduces verbatim the SSL protocol specification from <http://home.netspace.com/newsref/std/ssl.html>.

The SSL protocol is designed to establish a secure connection between a client and a server communicating over an insecure channel. This document makes several traditional assumptions, including that attackers have substantial computational resources and cannot obtain secret information from sources outside the protocol. Attackers are assumed to have the ability to capture, modify, delete, replay, and otherwise tamper with messages sent over the communication channel. The [following material] outlines how SSL has been designed to resist a variety of attacks.

Handshake Protocol

The handshake protocol is responsible for selecting a CipherSpec and generating a MasterSecret, which together comprise the primary cryptographic parameters associated with a secure session. The handshake protocol can also optionally authenticate parties who have certificates signed by a trusted certificate authority.

Authentication and Key Exchange

SSL supports three authentication modes: authentication of both parties, server authentication with an unauthenticated client, and total anonymity. Whenever the server is authenticated, the channel should be secure against man-in-the-middle attacks, but completely anonymous sessions are inherently vulnerable to such attacks. Anonymous servers cannot authenticate clients, since the client signature in the certificate verify message may require a server certificate to bind the signature to a particular server. If the server is authenticated, its certificate message

must provide a valid certificate chain leading to an acceptable certificate authority. Similarly, authenticated clients must supply an acceptable certificate to the server. Each party is responsible for verifying that the other's certificate is valid and has not expired or been revoked.

The general goal of the key exchange process is to create a `pre_master_secret` known to the communicating parties and not to attackers. The `pre_master_secret` will be used to generate the `master_secret`... . The `master_secret` is required to generate the finished messages, encryption keys, and MAC secrets... . By sending a correct finished message, parties thus prove that they know the correct `pre_master_secret`.

Anonymous key exchange

Completely anonymous sessions can be established using RSA, Diffie-Hellman, or Fortezza for key exchange. With anonymous RSA, the client encrypts a `pre_master_secret` with the server's uncertified public key extracted from the server key exchange message. The result is sent in a client key exchange message. Since eavesdroppers do not know the server's private key, it will be infeasible for them to decode the `pre_master_secret`.

With Diffie-Hellman or Fortezza, the server's public parameters are contained in the server key exchange message and the client's are sent in the client key exchange message. Eavesdroppers who do not know the private values should not be able to find the Diffie-Hellman result (i.e., the `pre_master_secret`) or the Fortezza token encryption key (TEK).

WARNING Completely anonymous connections only provide protection against passive eavesdropping. Unless an independent tamper-proof channel is used to verify that the finished messages were not replaced by an attacker, server authentication is required in environments where active man-in-the-middle attacks are a concern.

RSA key exchange and authentication

With RSA, key exchange and server authentication are combined. The public key may be either contained in the server's certificate or may be a temporary RSA key sent in a server key exchange message. When temporary RSA keys are used, they are signed by the server's RSA or DSS certificate. The signature includes the current `ClientHello.random`, so old signatures and temporary keys cannot be replayed. Servers may use a single temporary RSA key for multiple negotiation sessions.

NOTE The temporary RSA key option is useful if servers need large certificates but must comply with government-imposed size limits on keys used for key exchange.

After verifying the server's certificate, the client encrypts a `pre_master_secret` with the server's public key. By successfully decoding the `pre_master_secret` and producing a correct finished message, the server demonstrates that it knows the private key corresponding to the server certificate.

When RSA is used for key exchange, clients are authenticated using the certificate verify message (see Section 7.6.8). The client signs a value derived from the `master_secret` and all preceding handshake messages. These handshake messages include the server certificate, which binds the signature to the server, and `ServerHello.random`, which binds the signature to the current handshake process.

Diffie-Hellman key exchange with authentication

When Diffie-Hellman key exchange is used, the server can either supply a certificate containing fixed Diffie-Hellman parameters or can use the client key exchange message to send a set of temporary Diffie-Hellman parameters signed with a DSS or RSA certificate. Temporary parameters are hashed with the `hello.random` values before signing to ensure that attackers do not replay old parameters. In either case, the client can verify the certificate or signature to ensure that the parameters belong to the server.

If the client has a certificate containing fixed Diffie-Hellman parameters, its certificate contains the information required to complete the key exchange. Note that in this case the client and server will generate the same Diffie-Hellman result (i.e., `pre_master_secret`) every time they communicate. To prevent the `pre_master_secret` from staying in memory any longer than necessary, it should be converted into the `master_secret` as soon as possible. Client Diffie-Hellman parameters must be compatible with those supplied by the server for the key exchange to work.

If the client has a standard DSS or RSA certificate or is unauthenticated, it sends a set of temporary parameters to the server in the client key exchange message, then optionally uses a certificate verify message to authenticate itself.

Fortezza

Fortezza's design is classified, but at the protocol level it is similar to Diffie-Hellman with fixed public values contained in certificates. The result of the key exchange process is the token encryption key (TEK), which is used to wrap data encryption keys, client write key, server write key, and master secret encryption

key. The data encryption keys are not derived from the `pre_master_secret` because unwrapped keys are not accessible outside the token. The encrypted `pre_master_secret` is sent to the server in a client key exchange message.

Version Rollback Attacks

Because SSL Version 3.0 includes substantial improvements over SSL Version 2.0, attackers may try to make Version 3.0-capable clients and servers fall back to Version 2.0. This attack is occurring if (and only if) two Version 3.0-capable parties use an SSL 2.0 handshake.

Although the solution using non-random PKCS #1 block type 2 message padding is inelegant, it provides a reasonably secure way for Version 3.0 servers to detect the attack. This solution is not secure against attackers who can brute force the key and substitute a new ENCRYPTED-KEY-DATA message containing the same key (but with normal padding) before the application specified wait threshold has expired. Parties concerned about attacks of this scale should not be using 40-bit encryption keys anyway. Altering the padding of the least-significant 8 bytes of the PKCS padding does not impact security, since this is essentially equivalent to increasing the input block size by 8 bytes.

Detecting Attacks Against the Handshake Protocol

An attacker might try to influence the handshake exchange to make the parties select different encryption algorithms than they would normally choose. Because many implementations will support 40-bit exportable encryption and some may even support null encryption or MAC algorithms, this attack is of particular concern.

For this attack, an attacker must actively change one or more handshake messages. If this occurs, the client and server will compute different values for the handshake message hashes. As a result, the parties will not accept each others' finished messages. Without the `master_secret`, the attacker cannot repair the finished messages, so the attack will be discovered.

Resuming Sessions

When a connection is established by resuming a session, new `ClientHello.random` and `ServerHello.random` values are hashed with the session's `master_secret`. Provided that the `master_secret` has not been compromised and that the secure hash operations used to produce the encryption keys and MAC secrets are secure, the connection should be secure and effectively independent from previous connections. Attackers cannot use known encryption keys or MAC secrets to

compromise the `master_secret` without breaking the secure hash operations (which use both SHA and MD5).

Sessions cannot be resumed unless both the client and server agree. If either party suspects that the session may have been compromised, or that certificates may have expired or been revoked, it should force a full handshake. An upper limit of 24 hours is suggested for session ID lifetimes, since an attacker who obtains a `master_secret` may be able to impersonate the compromised party until the corresponding session ID is retired. Applications that may be run in relatively insecure environments should not write session IDs to stable storage.

MD5 and SHA

SSL uses hash functions very conservatively. Where possible, both MD5 and SHA are used in tandem to ensure that non-catastrophic flaws in one algorithm will not break the overall protocol.

Protecting Application Data

The `master_secret` is hashed with the `ClientHello.random` and `ServerHello.random` to produce unique data encryption keys and MAC secrets for each connection. Fortezza encryption keys are generated by the token, and are not derived from the `master_secret`.

Outgoing data is protected with a MAC before transmission. To prevent message replay or modification attacks, the MAC is computed from the MAC secret, the sequence number, the message length, the message contents, and two fixed character strings. The message type field is necessary to ensure that messages intended for one SSL Record Layer client are not redirected to another. The sequence number ensures that attempts to delete or reorder messages will be detected. Since sequence numbers are 64-bits long, they should never overflow. Messages from one party cannot be inserted into the other's output, since they use independent MAC secrets. Similarly, the server-write and client-write keys are independent so stream cipher keys are used only once.

If an attacker does break an encryption key, all messages encrypted with it can be read. Similarly, compromise of a MAC key can make message modification attacks possible. Because MACs are also encrypted, message-alteration attacks generally require breaking the encryption algorithm as well as the MAC.

NOTE MAC secrets may be larger than encryption keys, so messages can remain tamper resistant even if encryption keys are broken.

Final Notes

For SSL to be able to provide a secure connection, both the client and server systems, keys, and applications must be secure. In addition, the implementation must be free of security errors.

The system is only as strong as the weakest key exchange and authentication algorithm supported, and only trustworthy cryptographic functions should be used. Short public keys, 40-bit bulk encryption keys, and anonymous servers should be used with great caution. Implementations and users must be careful when deciding which certificates and certificate authorities are acceptable; a dishonest certificate authority can do tremendous damage.

Index

Symbols/Numbers

- # for comments, 16
- (hyphen), to set options, 65
- + (plus sign), to set options, 65

- 1 flag (kill command), 45
- 16 flag (kill command), 45

A

access

- anonymous, 85–88
- checking, 188–192
- to configuration and request information, 156
- controlling, 81, 149
- access.conf file, 20, 29
- AccessFileName directive, 48
- ACTION attribute (HTML), 58
- Action directive, 68
- actions, 68
- AddDescription directive, 100
- AddHandler directive, 67, 93
- AddIcon directive, 100
- AddIconByEncoding directive, 102
- AddIconByType directive, 101
- addresses
 - email, for automatic replies, 35
 - IP (see IP addresses)
 - loopback, 23
 - web, 10
- AddType directive, 93

adduser command, 22, 80

aliases

- alias command, 28
- Alias directive, 108
- ServerAlias directive, 35

All option, 64

allow from directive, 81

AllowOverride directive, 49–52

anonymous

- access, 85–88
- Anonymous directives, 86–89
- key exchange, 242

Apache

- API for, 152–174
- compiling, 12
- directives (see directives)
- history of, x
- loading, 15–19
- multiple copies of, 39–41
- technical support, 233
- versions of, ix, 13
- API, Apache, 152–174
- append_arrays function, 159
- array_cat function, 158
- arrays, 158
- asymmetric key encryption, 212
- authentication, 3, 75–77
 - digest, 82–85
 - directives for, 71–73, 77–82
 - modules for, 149
 - passwords, 73–75
 - SSL protocol and, 241–244

AuthGroupFile directive, 72
 AuthName directive, 72
 AuthType directive, 71, 83
 AuthUserFile directive, 72

B

BADMMAP rule, 18
 base URL, rewriting, 112
 bastion hosts, 215–218
 binary signatures, 209–214
 BindAddress directive, 40
 block_alarms function, 173
 block directives, 38
 blocking access, 149
 BrowserMatch directive, 62
 BrowserMatchNoCase directive, 62
 browsers, 62
 cookies and, 78
 HTTP/1.1 and, 95
 icons and, 100
 images and, 91
 languages and, 92
 bugs, 4
 keep-alives and Netscape, 37
 -X flag (httpd), 21
 bytes_in_free_blocks function, 164
 bytes_in_pool function, 164

C

-c flag (htpasswd), 74
 CA (certificate authority), 213–214
 CacheDefaultExpire directive, 125
 CacheDir... directives, 125
 CacheGcInterval directive, 125
 CacheLastModifiedFactor directive, 125
 CacheMaxExpire directive, 125
 CacheRoot directive, 124
 CacheSize directive, 125
 caching, 124–128
 call_exec function, 163
 can_exec function, 169
 “cannot determine local hostname”, 23
 carriage returns and line feeds (CRLF), 11
 CD-ROM with this book, xi
 cern_meta_module, 17
 certificates, 213–214
 cfg_getline function, 167

CGI (Common Gateway Interface), 4,
 53–55
 directives for, 55
 environment variables with, 61
 scripts as server-side includes, 130, 133
 cgi-script handler, 67
 cgi-bin directory, 4
 chdir_file function, 166
 child servers, limits on, 44
 chmod command, 26
 cipher suites, 231
 classes of networks, 6
 cleanup_for_exec function, 161
 cleanups, 160
 clear_pool function, 157
 clients, 10–11
 cmd command, 131
 cmd_how structure, 182
 command table, 182–185
 command_rec structure, 182
 comments in Configuration file, 16
 compiling Apache, 12
 compress utility, 14
 compression software, 13–15
 conf directory, 4, 36
 config command, 130
 config file
 httpd.conf, 20, 22
 inetd utility, 12
 server, 20
 configuration files, Apache, 15
 configuring
 Apache, 17
 modules, 153, 178–180
 proxy servers, 126–128
 construct_server function, 168
 construct_url function, 168
 content negotiation, 90–97
 Content-encoding header, 95
 Content-language header, 95
 Content-length header, 95
 Content-type header, 54, 95
 cookies, 78
 CookieTracking directive, 78
 copy_array function, 158
 copy_array_hdr function, 158
 copy_table function, 159
 “couldn’t determine user name”, 22

count_dirs function, 166
counters, 149
CRLF (carriage returns and line feeds), 11
cryptography (see encryption)

D

-d flag (httpd), 20, 25, 36
db_auth_module, 17
DBM files, 111
dbm_auth_module, 17
dbmmanage utility, 79
debugging, -X flag for (httpd), 21
decryption (see encryption)
deleting pools, 156
demonstration web sites, xi
deny from directive, 81
destroy_pool function, 156
destroy_sub_req function, 172
Diffie-Hellman key exchange, 243
digest authentication, 82–85
digital signatures, 209–214
directives, xiii, 35–43, 44, 48–52
 anonymous access, 86–89
 Apache-SSL, 229–232
 authentication, 71–73, 77–82
 browsers, 62
 caching, 125
 for CGI scripts, 55
 for environment variables, 61
 expiration, 96
 handlers, 67
 for HTTP headers, 64–67
 indexing, 99
 limiting application of, 72
 logging, 143–145
 overriding, 49–52
 proxy, 122–124
 rewriting, 110–114
 (see also under specific directive name)
directories
 execute permission for, 25
 iconifying, 100
 per-directory module
 configuration, 154, 179
 web site, 4
Directory directive, 38
DirectoryIndex directive, 93, 104–106
dld_module, 17

DMB files, 79–81
DNS, reverse lookup, 37
document root directory, 23
documentation
 AddDescription directive for, 100
 ReadmeName directive for, 102
 technical support, 233

E

echo command, 130, 134
echo.c program, 59–61
echo2.c program, 235–238
email address for automatic replies, 35
encoding, 95, 102
encryption, 210–212
 digest authentication, 82–85
 national security and, 220
encryption (see authentication)
env utility, 58
environment variables, 61, 130, 134
 browsers and, 62
error messages, 3
errors
 HTTP codes for, 146
 image maps, 117
 passwords and Anonymous
 directives, 88
 ServerAdmin directive, 35, 42
escape_html function, 168
escape_path_segment function, 168
escape_shell_cmd function, 167
escape_uri function, 168
/etc/hosts file, 28
/etc/inetd.conf file, 12, 46
exec command, 130, 133
ExecCGI option, 64–66
execute permission (see permissions)
expirations, 96, 125
 default time, 97
ExpiresActive directive, 96
ExpiresByType directive, 96
ExpiresDefault directive, 97
extensions, filename, 67, 93–95
external users, 206–208

F

-f flag (httpd), 25, 36
FancyIndexing directive, 99, 103

files

- DBM files, 79–81
- on enclosed CD-ROM, xii
- filename extensions, 67, 93–95
- filename functions, 165
- including in other, 131, 133
- indexing, 98–106
- limits on, 43
- permissions, 25–27
- size, 131, 132
- *.var (see type maps)

Files directive, 39

filters

- Action directive for, 68
- packet filtering, 215

find_token function, 167

firewalls, 214–218

fixed-length buffers, 204

flastmod command, 131, 132

FollowSymLinks option, 64, 67

FollowSymLinksIfOwnerMatch option, 67

format of log files, 144–147

formatted menus, 120

forms, 58

- authentication with, 75–77
- echo.c program, 59–61

Fortezza encryption keys, 243

FQDNs (fully qualified domain names), 28

FreeBSD UNIX, 13

freeware, 5

fsize command, 131, 132

functions, 156–174

G

get_client_block function, 171

get_gmtoff function, 164

get_local_host function, 170

get_time function, 164

get_token function, 167

get_virhost_addr function, 169

getparents function, 165

getword function, 166

getword_conf function, 167

getword_nulls function, 166

getword_white function, 166

gm_timestr_822 function, 164

gname2id function, 169

groups

- creating, 22
- permissions (see permissions)

gzip utility, 13–15

H

-h flag (httpd), 25

handlers, 67, 176, 195

handshake protocol, 241–245

hard_timeout function, 173

Header directive, 64

HeaderName directive, 102

headers

CGI, 54

HTTP response, 64–67

help, 233

history of Apache, x

Host header, 35

HostNameLookups directive, 37

hostnames, 10

- "cannot determine local hostname", 23
- controlling access from, 81
- mapping several to one address, 35
- reverse-DNS lookup, 37

hosts, 2

/etc/hosts file, 28

host numbers, 6

nonrouting (bastion), 215–218

virtual (see virtual hosts)

\windows\hosts file, 28

ht_time function, 164

.htaccess file, 47–52, 106

htdigest utility, 84

htdocs directory, 4

HTML (Hypertext Markup Language), 33

forms, 58

image maps, 118–121

htpasswd utility, 73

HTTP (Hypertext Transfer Protocol), 2

methods, 2, 72

response headers, 64–67

status codes, 146, 176–178

version 1.1 and browsers, 95

HTTP_ACCEPT variable, 91

HTTP_ACCEPT_LANGUAGE variable, 92

httpd, 4, 19

flags, 25

restarting, 45

virtual, number of, 43

httpd.conf file, 20, 22
hyphen (-), to set options, 65

I

icons, 100–102
IconsAreLinks option, 103
IDEA (International Data Encryption Algorithm), 212
identd daemon, 77
IdentityCheck directive, 77
ifconfig utility, 9
IfModule directive, 17
ignoring files in index, 99
image maps, 116–121
image negotiation, 91
imap-file handler, 67
ImapBase directive, 116
ImapDefault directive, 117
ImapMenu directive, 120
include command, 131, 133
includes (see server-side includes)
IncludesNoExec option, 65
index.html file, 34
Indexes option, 65
IndexIgnore directive, 99
indexing, 98–106
 icons with, 100–102
 image maps, 116–121
IndexOptions directive, 103
inetd mode (ServerType), 46
inetd utility, 12, 46
inetd.conf file, 12
initializer, 185
installing Apache, 15–19
interfaces, 8
internal users, 206–208
internal_redirect function, 172
internal_redirect_handler function, 172
International Data Encryption Algorithm (IDEA), 212
internationalization, modules for, 150
IP addresses, 6, 8
 binding to specific, 40
 controlling access from, 81
 loopback, 23
 mapping several hostnames to, 35
 restricting attention to, 40
IRIXNIS rule, 18
is_directory function, 169

is_matchexp function, 165
is_url function, 169
ISMAP attribute, 118

K

KeepAlive directive, 37, 63
keepalive_timeout function, 173
KeepAliveTimeout directive, 37
key escrow system, 221
key exchange, 241–244
keys, encryption (see encryption)
kill command, 24, 36
 -1 and -16 flags, 45
kill_cleanup function, 161
kill_cleanups_for_fd function, 161
kill_timeout function, 173

L

language negotiation, 91
languages, modules for, 150
legal issues, 218–221
level numbers, 94
license, Apache, x
Limit directive, 72
Listen directive, 40
ln command, 33, 66
loading Apache, 15–19
local networks, 27
Location directive, 39
Location header, 54
LogFormat directive, 144–147
logging, 143–147
 cache documents (see caching)
 CGI script information, 55
 logs directory, 4, 36
 module for, 196
 RewriteLog, RewriteLogLevel directives, 110
 TransferLog directive, 42
loopback addresses, 23

M

MAC (Message Authentication Code), 244–246
Mail Exchange (MX) records, 217
make_array function, 158
make_dirstr function, 166
make_full_path function, 169

- make_sub_pool function, 156
- make_table function, 159
- Makefile file, 15
- making Apache, 15–19
- MaxClients directive, 44
- MaxRequestsPerChild directive, 44
- MaxSpareServers, 44
- MD5 function, 83
- memory for pools, 157
- menus, 120
- merger modules, 180–182
- messages, error (see error messages)
- <METHOD> tag (HTML), 58
- methods, HTTP, 2, 72
- MIME types, 95, 192–194
 - icons by, 101
- MinSpareServers directive, 44
- modification time/date
 - cache, 125
 - expirations and, 96
 - flastmod command for, 131, 132
 - indexing, 103
- modules, 5, 16, 148, 175
 - authentication, 149
 - for blocking access, 149
 - configuring, 153
 - counter, 149
 - example of, 197–204
 - mod_access, 188
 - mod_alias, 107
 - mod_auth_anon, 85
 - mod_cgi, 107
 - mod_expires, 96
 - mod_include, 129–135
 - mod_rewrite, 109
 - mod_simultaneous, 150
 - mod_speling, 150
 - mod_status, 136–147
 - for rewriting URLs, 109, 150
 - structure of, 178–197
 - (see also under specific module name)
- mysql_auth_module, 17
- multiple copies of Apache, 39–41
- multitasking, 2
- multithreading, 204
- multiviews, 90–92
- MultiViews option, 65
- mutual-failure option, 81
- MX records, 217

N

- names
 - FQDNs, 28
 - hostnames (see hostnames)
- national security, 220
- NCSA server, 239
- negotiation, content, 90–97
- netmask command, 28
- Netscape, 30
 - cookies, 78
 - echo.c program, 59–61
 - keep-alive bug, 37
 - languages and, 92
- networks
 - classes of, 6
 - local, 27
 - network numbers, 6, 28
 - physically separate, 215–218
- no2slash function, 165
- NoCache directive, 126
- nokeepalive variable, 63
- nonce, 83
- nonrouting hosts, 215–218
- note_cleanups_for_fd function, 161
- note_cleanups_for_file function, 162
- note_subprocess function, 162
- numbers
 - host, 6
 - network, 6, 28
 - port, 8

O

- obtaining
 - Apache, 15
 - FreeBSD UNIX, 13
 - gzip utility, 13–15
- operating system, 18
- Options directive, 64
- order directive, 81
- os_escape_path function, 168
- overlay_tables function, 160
- overriding directives, 49–52

P

- packet filtering, 215
- pallocc function, 157
- parseHTTPdate function, 174
- PassEnv directive, 62

- passwords, 73–75
 - Anonymous directives and, 88
 - DBM files for, 79–81
- patents, 218
- pathnames, xiii, 10, 169
- paths, 35
- palloc function, 157
- pclosef function, 162
- performance
 - caching, 124–128
 - modules to improve, 150
 - PK encryption, 212
- permissions, 25–27
- per-server configuration, 153
- persistent-state cookies, 78
- pfclose function, 162
- pfopen function, 162
- pfopen function, 162
- PidFile directive, 36
- PIDs (process identifiers), 21, 36
- ping command, 28
- PK encryption, 210–212
- plus sign (+), to set options, 65
- pools, 152, 156
- popenf function, 161
- Port directive, 40
- ports, 2, 8, 40
- pregcomp function, 162
- pregsub function, 165
- privacy (see encryption; security)
- process identifiers (see PIDs)
- processes, killing, 24
- protocols, 8
- proxy servers, 3, 122–128
- ProxyPass directive, 123
- ProxyRemote directive, 123
- ProxyRequests directive, 123
- ps utility, 21
- pstrcat function, 157
- pstrdup function, 157
- pstrndup function, 157
- public key encryption, 210–212
- push_array function, 158

Q

- quality scores (qs values), 94

R

- read permission (see permissions)
- ReadmeName directive, 102
- realms, authentication, 72
- Redirect directive, 109
- redirection, 107–116, 172
- register_cleanup function, 160
- regular expressions for URLs, 109–116
- remote proxies, 123
- requests
 - maximum wait time, 37
 - per-request module configuration, 154
 - redirecting, 107–116
 - simultaneous, maximum for, 44
 - status information, 137
- require directive, 73
- reset_timeout function, 173
- response codes, HTTP, 146, 176–178
- response headers, 64–67
- restarting httpd, 45
- resuming sessions, 244
- reverse-DNS lookups, 37
- Rewrite module, 109–116
- RewriteBase directive, 112
- RewriteCond directive, 112
- RewriteEngine directive, 110
- RewriteLog directive, 110
- RewriteLogLevel directive, 111
- RewriteMap directive, 111
- RewriteRule directive, 113
- rewriting, modules for, 109, 150
- root directory, 23
- root user, 9, 22
- routers, 7
- rputc function, 170
- rputs function, 170
- RSA algorithm, 219, 242
- run_cleanup function, 170
- run_sub_req function, 172
- rvprintf function, 171
- rvputs function, 171

S

- ScanHTMLTitles option, 103
- ScriptAlias command, 107
- ScriptAlias directive, 55

- ScriptLog directive, 55
 - ScriptLogBuffer directive, 55
 - ScriptLogLength directive, 55
 - scripts, CGI (see CGI)
 - security, 3, 205–218
 - allow/deny from directives, 81
 - authentication (see authentication)
 - blocking access, 149
 - certificates, 213–214
 - cipher suites, 231
 - encryption, 210–212
 - file permissions, 25–27
 - firewalls, 214–218
 - fixed-length buffers, 204
 - handshake protocol and, 244
 - IgnoreIndex directive and, 99
 - legal issues, 218–221
 - national, 220
 - passwords, 73–75, 88
 - proxy servers, 122–128
 - SSL, 222–232
 - suEXEC program, 63
 - semiformatted menus, 120
 - send-as-is handler, 67
 - send_fd function, 170
 - send_fd_length function, 170
 - send_http_header function, 174
 - server-info handler, 67
 - server-parsed handler, 67
 - server-status handler, 67
 - ServerAdmin directive, 35, 42
 - ServerAlias directive, 35
 - ServerName directive, 35
 - ServerPath directive, 35
 - ServerRoot directive, 36
 - servers, 12
 - child, setting limits on, 44
 - config file, 20
 - getting information about, 136–143
 - maximum wait for requests, 37
 - per-server configuration, 153
 - per-server module configuration, 178
 - proxy, 3, 122–128
 - server-side includes, 129–135
 - IncludesNoExec option, 65
 - scripting modules, 150
 - ServerType directive, 46
 - sessions, resuming, 244
 - SetEnv directive, 61
 - SetHandler directive, 68
 - setup_client_block function, 171
 - shapes of image map hotspots, 119
 - should_client_block function, 171
 - Simple Mail Transfer Protocol (SMTP), 217
 - simultaneous requests, 44
 - size
 - cache, 125
 - files, 131, 132
 - pool, 164
 - SMTP (Simple Mail Transfer Protocol), 217
 - SOCKS rules, 17
 - soft_timeout function, 173
 - spawn_child function, 163
 - spawn_child_err function, 163
 - spelling module, 150
 - srn.conf file, 20, 29, 93
 - SSI (see server-side includes)
 - SSL protocol specification, 222–229, 241–246
 - directives for, 229–232
 - standalone mode, 12
 - standalone mode (ServerType), 46
 - StartServers directive, 45
 - status codes, HTTP, 146, 176–178
 - STATUS rule, 18
 - status, server, 136–143
 - strcasecmp_match function, 165
 - strcmp_match function, 165
 - strings
 - functions for, 165
 - in pools, 157
 - sub_req_lookup_file function, 172
 - sub_req_lookup_uri function, 172
 - subnet masks, 7
 - suEXEC program, 63
 - superuser, 9, 22
 - SuppressDescription option, 103
 - SuppressLastModified option, 103
 - SuppressSize option, 103
 - symbolic links, 33, 66
 - SymLinksIfOwnerMatch option, 65
- ## T
- tables
 - command table, 182–185
 - table _ functions for, 159–160
 - TCP (Transmission Control Protocol), 7
 - TCP/IP, 5–9

technical support, 233

telnet, 11

time

 caching-related, 125

 expirations, 96

 functions for, 164

TimeOut directive, 37

timeouts, functions for, 173

tm2sec function, 174

token encryption key (TEK), 243

TransferLog directive, 42, 143

translating URLs to names, 186

type checker, 192–194

type-map handler, 67

type maps, 93–95

U

UDP (User Datagram Protocol), 7

uname2id function, 169

unblock_alarms function, 174

unescape_url function, 168

unformatted menus, 121

UNIX versions, 13

URIs (uniform resource identifiers), 1

URLs (uniform resource locators), 1, 10

 digest authentication and, 82–85

 functions for, 168

 image maps and, 116

 limiting Apache directives to, 39

 rewriting, 109–116

 translating to names, 186

user agents, 82

User-Agent header, 82

users

 checking if access allowed, 190–192

 “couldn’t determine user name”, 22

 creating, 22

 DBM files, 79–81

 identifying, 77

 permissions (see permissions)

 security and, 206–208

 validating, 209

uudecode function, 170

V

-v flag (httpd), 25

validating users, 209

*.var files (see type maps)

variables

 environment, 61

 printing values of, 130, 134

version rollback attacks, 244

versions

 Apache, ix, 13

 SOCKS, 17

 UNIX, 13

virtual cash, 209–214

virtual hosts, 7, 30, 41–45

 HTTP/1.1 and, 46

VirtualHost directive, 38

W

WANTHSREGEX rule, 17

web addresses (see URLs)

web sites, 4

 demonstration, xi

Windows 95, 28

\\windows\\hosts file, 28

write permissions (see permissions)

X

-X flag (httpd), 21, 25, 43

XBitHack facility, 134

XSSI facility, 134

About the Authors

Ben Laurie is a member of the core Apache Group and has made his living as a programmer since 1978. Peter Laurie, Ben's father, is a freelance journalist who has written several computer books. He is a former editor of *Practical Computing* magazine.

Colophon

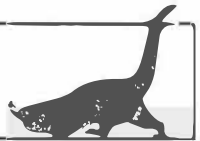
The animal featured on the cover of *Apache: The Definitive Guide* is an Appaloosa horse. Developed by the Nez Perce Indians of northeastern Oregon, the name Appaloosa derives from the nearby Palouse River. Although spotted horses are believed to be almost as old as the equine race itself—Cro-Magnon cave paintings depict spotted horses—the Appaloosa is the only established breed of spotted horse. The Appaloosa was bred to be a hunting and war horse, and as such they have great stamina, are highly athletic and agile, and have docile temperaments. When the Nez Perce, led by Chief Joseph, surrendered to the U.S. Army in 1876 and were exiled to Oklahoma, the Appaloosa breed was almost eradicated. In 1938 the Appaloosa Horse Club was formed in Moscow, Idaho, and the breed was revived. The Horse Club now registers approximately 65,000 horses, making it the third largest registry in the world. No longer a war horse, Appaloosas can be found in many equestrian venues, from trail riding to western competition to pleasure riding.

Edie Freedman designed the cover of this book, using a 19th-century engraving from the Dover Pictorial Archive. The cover layout was produced with Quark XPress 3.3 using the ITC Garamond font.

The inside layout was designed by Nancy Priest and implemented in FrameMaker by Mike Sierra. The text and heading fonts are ITC Garamond Light and Garamond Book. The illustrations that appear in the book were created in Macromedia Freehand 5.0 by Chris Reilley. This colophon was written by Clairemarie Fisher O'Leary.



More Titles from O'Reilly



Web Server Administration

Apache: The Definitive Guide



By Ben Laurie & Peter Laurie
1st Edition Winter 1997
300 pages (est.), ISBN 1-56592-250-6

Despite all the hype about Netscape, Apache is far and away the most widely used Web server platform in the world. It runs on about half the world's existing Web sites and is rapidly increasing market share. *Apache: The Definitive Guide* is the only complete guide on the market today describing how to obtain, set up, and secure the Apache software. Officially authorized by the Apache Group, this book is the definitive documentation for the world's most popular Web server.

Contents include:

- The history of the Apache Group
- Obtaining and compiling the server
- Configuring and running Apache, including such topics as directory structures, virtual hosts, and CGI programming
- The Apache Module API
- Apache security
- The Apache manuals
- A complete list of configuration directives
- A complete demo of a sample Web site

UNIX Web Server Administration



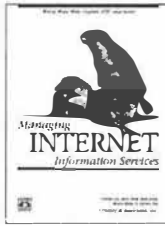
By John Leavitt
1st Edition Winter 1997
325 pages (est.), ISBN 1-56592-217-4

With our increasing dependence on Web sites for our daily work, the Web server has emerged as one of the most crucial services a company can offer. When the server is slow, customers are frustrated. When the server or network is down,

customers are turned away.

UNIX Web Server Administration tells Web administrators how to keep a server running smoothly. This book not only covers Apache, the most popular server on the Internet, but also the NCSA, CERN, and Netscape servers.

Managing Internet Information Services



By Cricket Liu, Jerry Peek, Russ Jones,
Bryan Buus & Adrian Nye
1st Edition December 1994
668 pages, ISBN 1-56592-062-7

Managing Internet Information Services describes how to create services for the millions of Internet users. By setting up Internet servers for World Wide Web, Gopher, FTP, Finger, Telnet, WAIS (Wide Area Information Services), or email services, anyone with a suitable computer and Internet connection can become an "Internet Publisher."

Services on the Internet allow almost instant distribution and frequent updates of any kind of information. You can provide services to employees of your own company (solving the information distribution problems of spread-out companies), or you can serve the world. Perhaps you'd like to create an Internet service equivalent to the telephone company's directory assistance. Or maybe you're the Species Survival Commission, and you'd like your plans online; this book describes a prototype service the authors created to make SSC's endangered species Action Plans viewable worldwide. Whatever you have in mind can be done. This book tells you how.

Creating a service can be a big job, involving more than one person. This book separates the setup and maintenance of server software from data management, so that a team can divide responsibilities. Sections and chapters on data management, a role we call the Data Librarian, are marked with a special icon.

"Excellent book . . . carefully written, informative and readable . . . [well] organized. . . I'm enjoying it considerably. I picked it up to flip through—I'm a writer of Internet books myself, and I do a lot of 'been there, done that' when I approach an Internet book—and after an hour or so, I discovered I was actually reading the book. I have even taken the step of . . . using my post-it-notes tabs to mark the good stuff."

—Jill Ellsworth, Ph.D., Author of *The Internet Business Book*, *Education on the Internet*, and *Marketing on the Internet*

O'REILLY™

TO ORDER: 800-998-9938 • order@ora.com • <http://www.ora.com/>

OUR PRODUCTS ARE AVAILABLE AT A BOOKSTORE OR SOFTWARE STORE NEAR YOU.

FOR INFORMATION: 800-998-9938 070-829-0515 • info@ora.com

Developing Web Content

Building Your Own WebSite



By Susan B. Peck & Stephen Arrants
1st Edition July 1996
514 pages, ISBN 1-56592-232-8

This is a hands-on reference for Windows® 95 and Windows NT™ desktop users who want to host their own site on the Web or on a corporate intranet. You'll also learn how to connect your web to information in other Windows applications, such as word processing documents and databases. Packed

with examples and tutorials on every aspect of Web management. Includes the highly acclaimed WebSite™ 1.1 on CD-ROM.

Web Client Programming with Perl



By Clinton Wong
1st Edition March 1997
250 pages, ISBN 1-56592-214-X

Web Client Programming with Perl teaches you how to extend scripting skills to the Web. This book teaches you the basics of how browsers communicate with servers and how to write your own customized Web clients to automate common tasks. It is intended for those who are motivated to

develop software that offers a more flexible and dynamic response than a standard Web browser.

JavaScript: The Definitive Guide, Regular Edition

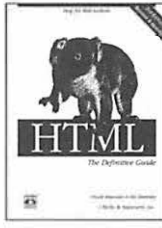


By David Flanagan
2nd Edition January 1997
672 pages, ISBN 1-56592-234-4

In this second edition, the author of the best-selling *Java in a Nutshell* describes the server-side JavaScript application, LiveWire, developed by Netscape and Sun Microsystems.

The book describes the version of JavaScript shipped with Navigator 2.0, 2.0.1, and 2.0.2, and also the much-changed version of JavaScript shipped with Navigator 3.0. LiveConnect, used for communication between JavaScript and Java applets, and addresses commonly encountered bugs on JavaScript objects.

HTML: The Definitive Guide, Second Edition



By Chuck Musciano & Bill Kennedy
2nd Edition April 1997
520 pages, ISBN 1-56592-235-2

The second edition covers the most up-to-date version of the HTML standard (the proposed HTML version 3.2), Netscape 4.0 and Internet Explorer 3.0, plus all the common extensions, especially Netscape extensions.

The authors address all the current version's elements, explaining how they work and interact with each other. Includes a style guide that helps you to use HTML to accomplish a variety of tasks, from simple online documentation to complex marketing and sales presentations.

Designing for the Web: Getting Started in a New Medium

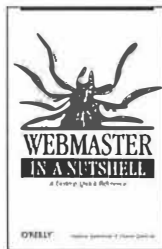


By Jennifer Niederst with Edie Freedman
1st Edition April 1996
180 pages, ISBN 1-56592-165-8

Designing for the Web gives you the basics you need to hit the ground running. Although geared toward designers, it covers information and techniques useful to anyone who wants to put graphics online. It explains how to work with HTML documents from a designer's point of view, outlines special problems with presenting information online, and walks through incorporating images into Web pages, with emphasis on resolution and improving efficiency.

WebMaster in a Nutshell

By Stephen Spainhour & Valerie Quercia



1st Edition October 1996
378 pages, ISBN 1-56592-229-8

Web content providers and administrators have many sources of information, both in print and online. *WebMaster in a Nutshell* pulls it all together into one slim volume – for easy desktop access. This quick-reference covers HTML, CGI, Perl, HTTP, server configuration, and tools for Web administration.

O'REILLY™

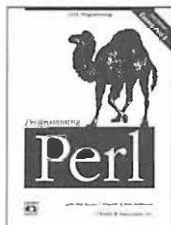
TO ORDER: 800-998-9938 • order@ora.com • <http://www.ora.com/>

OUR PRODUCTS ARE AVAILABLE AT A BOOKSTORE OR SOFTWARE STORE NEAR YOU.

FOR INFORMATION: 800-998-9938 • 800-729-0515 • info@ora.com

Perl

Programming Perl, Second Edition



By Larry Wall, Tom Christiansen,
& Randal L. Schwartz
2nd Edition September 1996
676 pages, ISBN 1-56592-149-6

Programming Perl, Second Edition, is coauthored by Larry Wall, the creator of Perl. Perl is a language for easily manipulating text, files, and processes. It provides a more concise and readable way to do

many jobs that were formerly accomplished (with difficulty) by programming with C or one of the shells. This heavily revised second edition contains a full explanation of Perl version 5.003.

Learning Perl, Second Edition



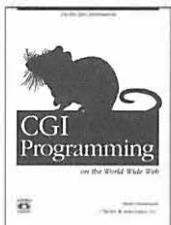
By Randal L. Schwartz
Foreword by Larry Wall
2nd Edition July 1997
400 pages, ISBN 1-56592-284-0

This second edition of *Learning Perl*, with a foreword by Perl author Larry Wall, fully covers Perl, Version 5. In this new edition, program examples and exercise answers have been radically updated to reflect typical usage under Perl 5, and

numerous details have been added or modified. In addition, you'll find new sections introducing Perl references and CGI programming.

Learning Perl, Second Edition is ideal for system administrators, programmers, and anyone else wanting a down-to-earth introduction to this useful language. Written by a Perl trainer, its aim is to make a competent, hands-on Perl programmer out of the reader as quickly as possible. The book takes a tutorial approach and includes hundreds of short code examples, along with some lengthy ones. The relatively inexperienced programmer will find *Learning Perl* easily accessible. For a comprehensive and detailed guide to advanced programming with Perl, read O'Reilly's companion book, *Programming Perl, Second Edition*.

CGI Programming on the World Wide Web



By Shishir Gundavaram
1st Edition March 1996
450 pages, ISBN 1-56592-168-2

This book offers a comprehensive explanation of CGI and related techniques for people who hold on to the dream of providing their own information servers on the Web. It starts at the beginning, explaining the value of CGI and how it works, then moves swiftly into the subtle details of programming.

Perl 5 Desktop Reference



By Johan Vromans
1st Edition February 1996
44 pages, ISBN 1-56592-187-9

This is the standard quick-reference guide for the Perl programming language. It provides a complete overview of the language, from variables to input and output, from flow control to regular expressions, from functions to document formats—all packed into a convenient, carry-around booklet. Updated to cover Perl version 5.003.

Mastering Regular Expressions



By Jeffrey E. F. Friedl
1st Edition January 1997
368 pages, ISBN 1-56592-257-3

Regular expressions, a powerful tool for manipulating text and data, are found in scripting languages, editors, programming environments, and specialized tools. In this book, author Jeffrey Friedl leads you through the steps of crafting a regular expression that gets the job done. He examines a variety of tools and uses them in an extensive array of examples, dedicating an entire chapter to Perl.

O'REILLY™

TO ORDER: 800-998-9938 • order@ora.com • <http://www.ora.com/>

OUR PRODUCTS ARE AVAILABLE AT A BOOKSTORE OR SOFTWARE STORE NEAR YOU.

FOR INFORMATION: 800-998-9938 0800-829-0515 • info@ora.com

How to stay in touch with O'Reilly

1. Visit Our Award-Winning Web Site

<http://www.ora.com/>

- ★ "Top 100 Sites on the Web" — *PC Magazine*
- ★ "Top 5% Web sites" — *Point Communications*
- ★ "3-Star site" — *The McKinley Group*

Our web site contains a library of comprehensive product information (including book excerpts and tables of contents), downloadable software, background articles, interviews with technology leaders, links to relevant sites, book cover art, and more. File us in your Bookmarks or Hotlist!

2. Join Our Email Mailing Lists

New Product Releases

To receive automatic email with brief descriptions of all new O'Reilly products as they are released, send email to: listproc@online.ora.com

Put the following information in the first line of your message (*not* in the Subject field):

subscribe ora-news "Your Name" of "Your Organization" (for example: subscribe ora-news Kris Webber of Fine Enterprises)

O'Reilly Events

If you'd also like us to send information about trade show events, special promotions, and other O'Reilly events, send email to: listproc@online.ora.com

Put the following information in the first line of your message (*not* in the Subject field):

subscribe ora-events "Your Name" of "Your Organization"

3. Get Examples from Our Books via FTP

There are two ways to access an archive of example files from our books:

Regular FTP

- ftp to:
[ftp.ora.com](ftp://ftp.ora.com)
(login: anonymous
password: your email address)
- Point your web browser to:
<http://ftp.ora.com/>

FTPMAIL

- Send an email message to:
ftpmail@online.ora.com
(Write "help" in the message body)

4. Visit Our Gopher Site

- Connect your gopher to:
[gopher.ora.com](gopher://gopher.ora.com)
- Point your web browser to:
gopher://gopher.ora.com/
- Telnet to:
<gopher.ora.com>
login: gopher

5. Contact Us via Email

order@ora.com

To place a book or software order online. Good for North American and international customers.

subscriptions@ora.com

To place an order for any of our newsletters or periodicals.

books@ora.com

General questions about any of our books.

software@ora.com

For general questions and product information about our software. Check out O'Reilly Software Online at <http://software.ora.com/> for software and technical support information. Registered O'Reilly software users send your questions to: website-support@ora.com

cs@ora.com

For answers to problems regarding your order or our products.

booktech@ora.com

For book content technical questions or corrections.

proposals@ora.com

To submit new book or software proposals to our editors and product managers.

international@ora.com

For information about our international distributors or translation queries. For a list of our distributors outside of North America check out:
<http://www.ora.com/www/order/country.html>

O'Reilly & Associates, Inc.

101 Morris Street, Sebastopol, CA 95472 USA

TEL 707-829-0515 or 800-998-9938

(6am to 5pm PST)

FAX 707-829-0104

O'REILLY™

TO ORDER: 800-998-9938 • order@ora.com • <http://www.ora.com/>

OUR PRODUCTS ARE AVAILABLE AT A BOOKSTORE OR SOFTWARE STORE NEAR YOU.

FOR INFORMATION: 800-998-9938 • 707-829-0515 • info@ora.com

Titles from O'Reilly

Please note that upcoming titles are displayed in *italic*.

WEB PROGRAMMING

Apache: The Definitive Guide
Building Your Own Web Conferences
Building Your Own Website
CGI Programming for the World Wide Web
Designing for the Web
HTML: The Definitive Guide, 2nd Ed.
JavaScript: The Definitive Guide, 2nd Ed.
Learning Perl
Programming Perl, 2nd Ed.
Mastering Regular Expressions
WebMaster in a Nutshell
Web Security & Commerce
Web Client Programming with Perl
World Wide Web Journal

USING THE INTERNET

Smileys
The Future Does Not Compute
The Whole Internet User's Guide & Catalog
The Whole Internet for Win 95
Using Email Effectively
Bandits on the Information Superhighway

JAVA SERIES

Exploring Java
Java AWT Reference
Java Fundamental Classes Reference
Java in a Nutshell
Java Language Reference, 2nd Edition
Java Network Programming
Java Threads
Java Virtual Machine

SOFTWARE

WebSite™ 1.1
WebSite Professional™
Building Your Own Web Conferences
WebBoard™
PolyForm™
Statisphere™

SONGLINE GUIDES

NetActivism NetResearch
Net Law NetSuccess
NetLearning NetTravel
Net Lessons

SYSTEM ADMINISTRATION

Building Internet Firewalls
Computer Crime: A Crimelfighter's Handbook
Computer Security Basics
DNS and BIND, 2nd Ed.
Essential System Administration, 2nd Ed.
Getting Connected: The Internet at 56K and Up
Linux Network Administrator's Guide
Managing Internet Information Services
Managing NFS and NIS
Networking Personal Computers with TCP/IP
Practical UNIX & Internet Security, 2nd Ed.
PGP: Pretty Good Privacy sendmail, 2nd Ed.
sendmail Desktop Reference
System Performance Tuning
TCP/IP Network Administration
termcap & terminfo
Using & Managing UUCP
Volume 8: X Window System Administrator's Guide
Web Security & Commerce

UNIX

Exploring Expect
Learning VBScript
Learning GNU Emacs, 2nd Ed.
Learning the bash Shell
Learning the Korn Shell
Learning the UNIX Operating System
Learning the vi Editor
Linux in a Nutshell
Making TeX Work
Linux Multimedia Guide
Running Linux, 2nd Ed.
SCO UNIX in a Nutshell
sed & awk, 2nd Edition
Tcl/Tk Tools
UNIX in a Nutshell: System V Edition
UNIX Power Tools
Using csh & tsh
When You Can't Find Your UNIX System Administrator
Writing GNU Emacs Extensions

WEB REVIEW STUDIO SERIES

Gif Animation Studio
Shockwave Studio

WINDOWS

Dictionary of PC Hardware and Data Communications Terms
Inside the Windows 95 Registry
Inside the Windows 95 File System
Windows Annoyances
Windows NT File System Internals
Windows NT in a Nutshell

PROGRAMMING

Advanced Oracle PL/SQL Programming
Applying RCS and SCCS
C++: The Core Language
Checking C Programs with lint
DCE Security Programming
Distributing Applications Across DCE & Windows NT
Encyclopedia of Graphics File Formats, 2nd Ed.
Guide to Writing DCE Applications
lex & yacc
Managing Projects with make
Mastering Oracle Power Objects
Oracle Design: The Definitive Guide
Oracle Performance Tuning, 2nd Ed.
Oracle PL/SQL Programming
Porting UNIX Software
POSIX Programmer's Guide
POSIX.4: Programming for the Real World
Power Programming with RPC
Practical C Programming
Practical C++ Programming
Programming Python
Programming with curses
Programming with GNU Software
Pthreads Programming
Software Portability with imake, 2nd Ed.
Understanding DCE
Understanding Japanese Information Processing
UNIX Systems Programming for SVR4

BERKELEY 4.4 SOFTWARE DISTRIBUTION

4.4BSD System Manager's Manual
4.4BSD User's Reference Manual
4.4BSD User's Supplementary Documents
4.4BSD Programmer's Reference Manual
4.4BSD Programmer's Supplementary Documents
X Programming
Vol. 0: X Protocol Reference Manual
Vol. 1: Xlib Programming Manual
Vol. 2: Xlib Reference Manual
Vol. 3M: X Window System User's Guide, Motif Edition
Vol. 4M: X Toolkit Intrinsics Programming Manual, Motif Edition
Vol. 5: X Toolkit Intrinsics Reference Manual
Vol. 6A: Motif Programming Manual
Vol. 6B: Motif Reference Manual
Vol. 6C: Motif Tools
Vol. 8: X Window System Administrator's Guide
Programmer's Supplement for Release 6
X User Tools
The X Window System in a Nutshell

CAREER & BUSINESS

Building a Successful Software Business
The Computer User's Survival Guide
Love Your Job!
Electronic Publishing on CD-ROM

TRAVEL

Travelers' Tales: Brazil
Travelers' Tales: Food
Travelers' Tales: France
Travelers' Tales: Gutsy Women
Travelers' Tales: India
Travelers' Tales: Mexico
Travelers' Tales: Paris
Travelers' Tales: San Francisco
Travelers' Tales: Spain
Travelers' Tales: Thailand
Travelers' Tales: A Woman's World

O'REILLY™

TO ORDER: 800-998-9938 • order@ora.com • <http://www.ora.com/>

OUR PRODUCTS ARE AVAILABLE AT A BOOKSTORE OR SOFTWARE STORE NEAR YOU.

FOR INFORMATION: 800-998-9938 • 607-829-0515 • info@ora.com

International Distributors

UK, Europe, Middle East and Northern Africa (except France, Germany, Switzerland, & Austria)

INQUIRIES

International Thomson Publishing
Europe
Berkshire House
168-173 High Holborn
London WC1V 7AA, United Kingdom
Telephone: 44-171-497-1422
Fax: 44-171-497-1426
Email: itpint@itps.co.uk

ORDERS

International Thomson Publishing
Services, Ltd.
Cheriton House, North Way
Andover, Hampshire SP10 5BE,
United Kingdom
Telephone: 44-264-342-832
(UK orders)
Telephone: 44-264-342-806
(outside UK)
Fax: 44-264-364418 (UK orders)
Fax: 44-264-342761 (outside UK)
UK & Eire orders: itpuk@itps.co.uk
International orders: itpint@itps.co.uk

France

Editions Eyrolles
61 bd Saint-Germain
75240 Paris Cedex 05
France
Fax: 33-01-44-41-11-44

FRENCH LANGUAGE BOOKS

All countries except Canada
Phone: 33-01-44-41-46-16
Email: geodif@eyrolles.com

ENGLISH LANGUAGE BOOKS

Phone: 33-01-44-41-11-87
Email: distribution@eyrolles.com

Australia

WoodsLane Pty. Ltd.
7/5 Vuko Place, Warriewood NSW 2102
P.O. Box 935, Mona Vale NSW 2103
Australia
Telephone: 61-2-9970-5111
Fax: 61-2-9970-5002
Email: info@woodslane.com.au

Germany, Switzerland, and Austria

INQUIRIES

O'Reilly Verlag
Balthasarstr. 81
D-50670 Köln
Germany
Telephone: 49-221-97-31-60-0
Fax: 49-221-97-31-60-8
Email: anfragen@oreilly.de

ORDERS

International Thomson Publishing
Königswinterer Straße 418
53227 Bonn, Germany
Telephone: 49-228-97024 0
Fax: 49-228-441342
Email: order@oreilly.de

Asia (except Japan & India)

INQUIRIES

International Thomson Publishing Asia
60 Albert Street #15-01
Albert Complex
Singapore 189969
Telephone: 65-336-6411
Fax: 65-336-7411

ORDERS

Telephone: 65-336-6411
Fax: 65-334-1617
thomson@signet.com.sg

New Zealand

WoodsLane New Zealand Ltd.
21 Cooks Street (P.O. Box 575)
Wanganui, New Zealand
Telephone: 64-6-347-6543
Fax: 64-6-345-4840
Email: info@woodslane.com.au

Japan

O'Reilly Japan, Inc.
Kiyoshige Building 2F
12-Banchi, Sanei-cho
Shinjuku-ku
Tokyo 160 Japan
Telephone: 81-3-3356-5227
Fax: 81-3-3356-5261
Email: kenji@ora.com

India

Computer Bookshop (India) PVT. LTD.
190 Dr. D.N. Road, Fort
Bombay 400 001
India
Telephone: 91-22-207-0989
Fax: 91-22-262-3551
Email: cbsbom@giabom01.vsnl.net.in

The Americas

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472 U.S.A.
Telephone: 707-829-0515
Telephone: 800-998-9938 (U.S. &
Canada)
Fax: 707-829-0104
Email: order@ora.com

Southern Africa

International Thomson Publishing
Southern Africa
Building 18, Constantia Park
138 Sixteenth Road
P.O. Box 2459
Halfway House, 1685 South Africa
Telephone: 27-11-805-4819
Fax: 27-11-805-3648

O'REILLY™

TO ORDER: 800-998-9938 • order@ora.com • <http://www.ora.com/>

OUR PRODUCTS ARE AVAILABLE AT A BOOKSTORE OR SOFTWARE STORE NEAR YOU.

FOR INFORMATION: 800-998-9938 • 707-829-0515 • info@ora.com



**CD-ROM
INCLUDED**

Architectural Drawing The Guideline

Laurie & Laurie

O'REILLY



Apache: The Definitive Guide



Apache is far and away the most widely used web server platform in the world. Freely available on the Internet, this versatile server runs on nearly half of the world's existing web sites and is rapidly increasing in market share.

With *Apache: The Definitive Guide*, web administrators new to Apache will come up to speed more quickly than ever before by working through the tutorial demo. Experienced administrators and CGI programmers will find the clear, concise reference sections indispensable, and system programmers interested in customizing their servers will turn immediately to the chapters on the API and Apache modules. Written and reviewed by members of the Apache Group, the core developers of the Apache server, *Apache: The Definitive Guide* is essential documentation for the world's most popular web server.

Contents include:

- The history of the Apache Group
- Downloading and compiling the server
- Configuring and running Apache on your UNIX machine
- Writing specialized Apache modules
- Security—Firewalls and SSL—in Apache

The book also includes a CD-ROM containing V1.1.1 and V1.2 beta of the Apache server software, complete system documentation, the HTTP/1.0 and HTTP/1.1 specifications, the CGI/1.1 specification, and all of the sample web sites used in the book.

Ben Laurie has been working as a programmer since 1978, and is a member of the core Apache Group. Peter Laurie is a freelance journalist and the author of several computer books.

"Another Nutshell book that is a 'must have' for Internet professionals using the Apache web server software. The authors have applied their ample qualifications to produce a handbook with enough information to remain a useful tool for the basics and beyond."

—Randy Terbush, *Apache Group member*



Printed on Recycled Paper

ISBN 1-56592-250-6